Model-Based Testing: An Introductory Tutorial

Shuai Wang, Senior Test Consultant

Agenda

- Introduction (30 min)
- Start MBT with a simple example (30 min)
- Short break (10 min)
- The value of models (40 min): three real case studies
- Q&A (10 min)

Agenda

- Introduction (30 min)
  - The speaker
  - What is MBT?
  - Why is MBT?
  - MBT tools

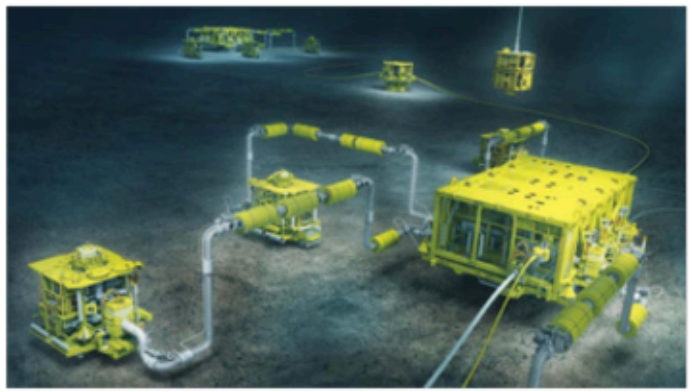
My self…

- Ph.D. within software engineering, Software V&V
  - Model-based software engineering
  - Search-based software engineering
  - Empirical software engineering
  - Evolutionary computation
- 7-year model-based research experience with many industrial partners
  - Cisco system (model-based testing)
  - Kreftregisteret (model-based data checking)
- More than 40 publications (I bet you don't care :D)
- *First time to give the tutorial in such a forum*

Focus of this tutorial

- Understanding of basic MBT principles
- Illustrations of MBT with an example
  - Test requirements to models
  - Models to test cases (tests)
- Benefits of using models in practice
  - Not only for testing!

Our lives are dependent on different large-scale systems



Subsea production system

Car

Airplane

Video conferencing system

Robot

Remote surgery

A bad quality software can put our lives in danger

Testing large-scale system is very expensive and time consuming

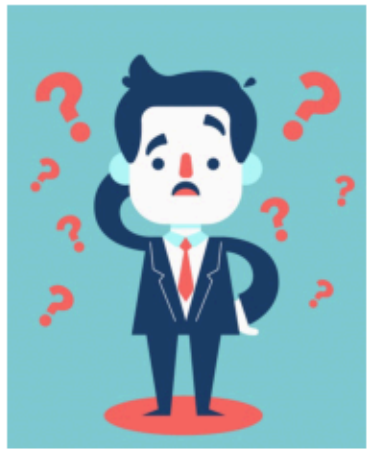
Diverse hardware

Diverse software

How to position MBT?

Test requirements → **Test design** → **Tests** → **Test execution**

Test Analyst



Manual design?

Tester

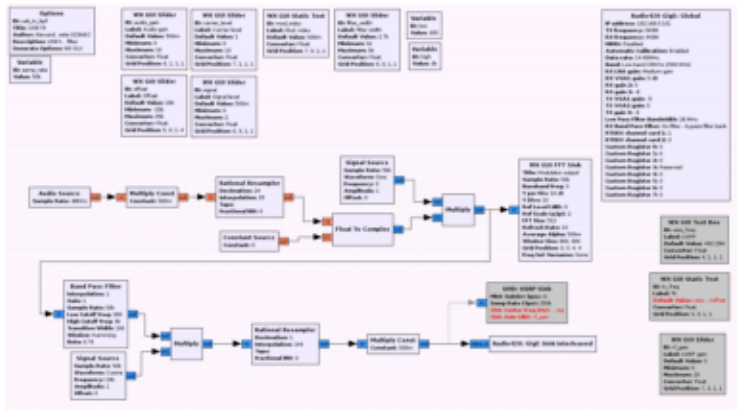
Tester

How to position MBT?

Test Requirements → **Designs** → **Tests** → **Test execution**

Test Analyst

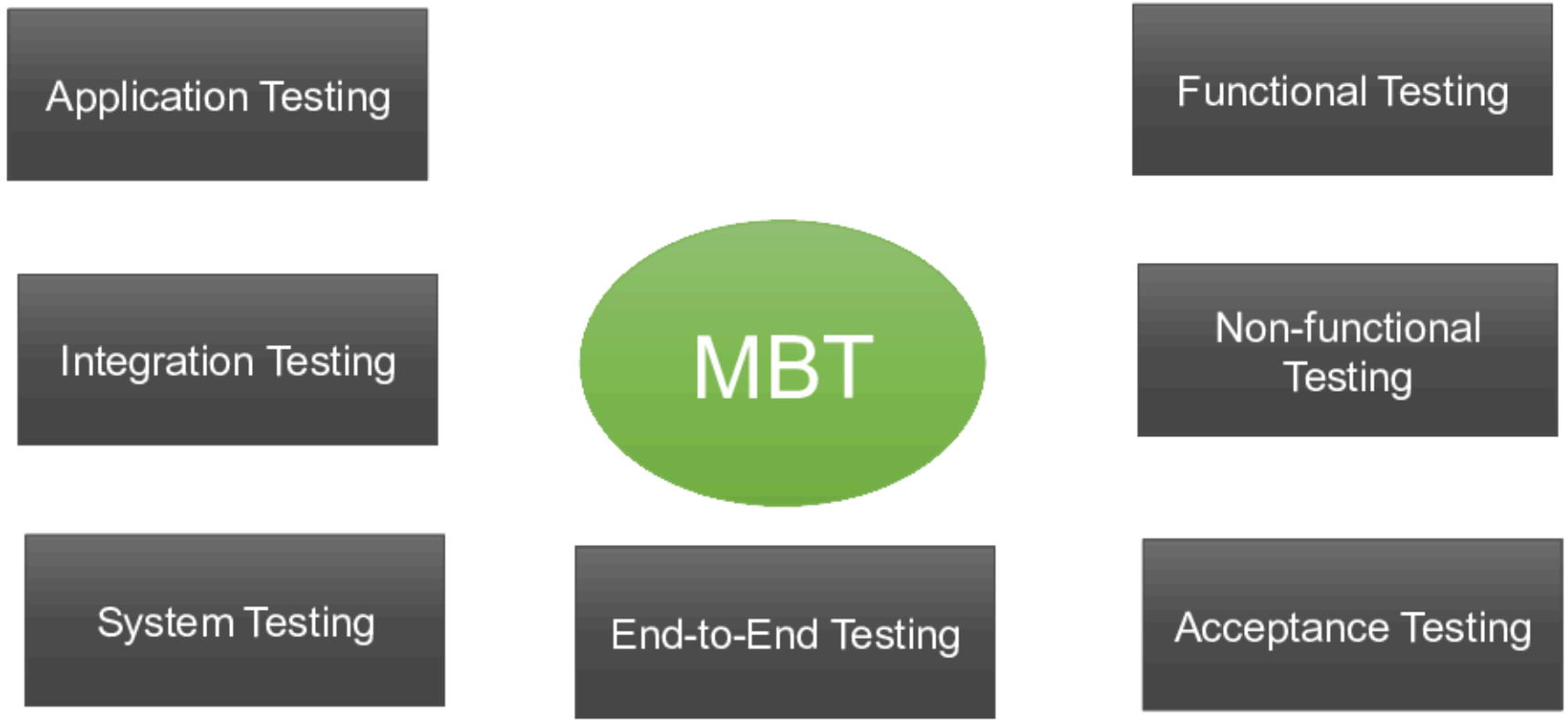
Tester



Model-based solutions

Tester

What kind of testing?



MBT definition

- Wikipedia:

Model-based testing is an application of model-based design for designing and optionally also executing artifacts to perform software testing or system testing. Models can be used to represent the desired behavior of a system under test (SUT), or to represent testing strategies and a test environment.

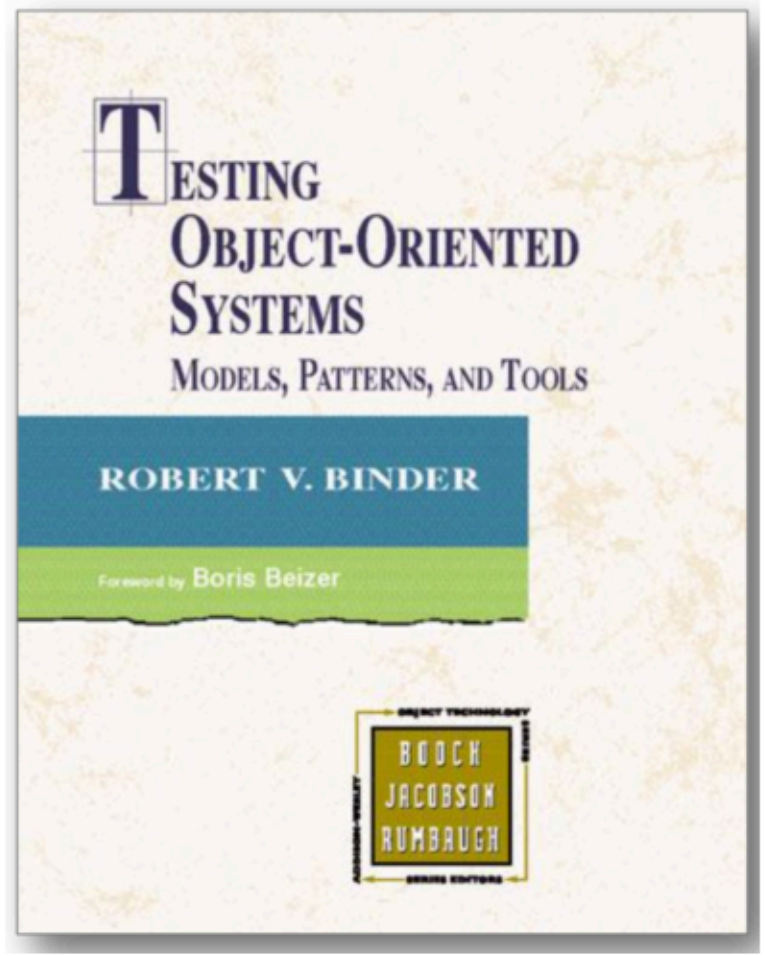
MBT definition

- Wikipedia:

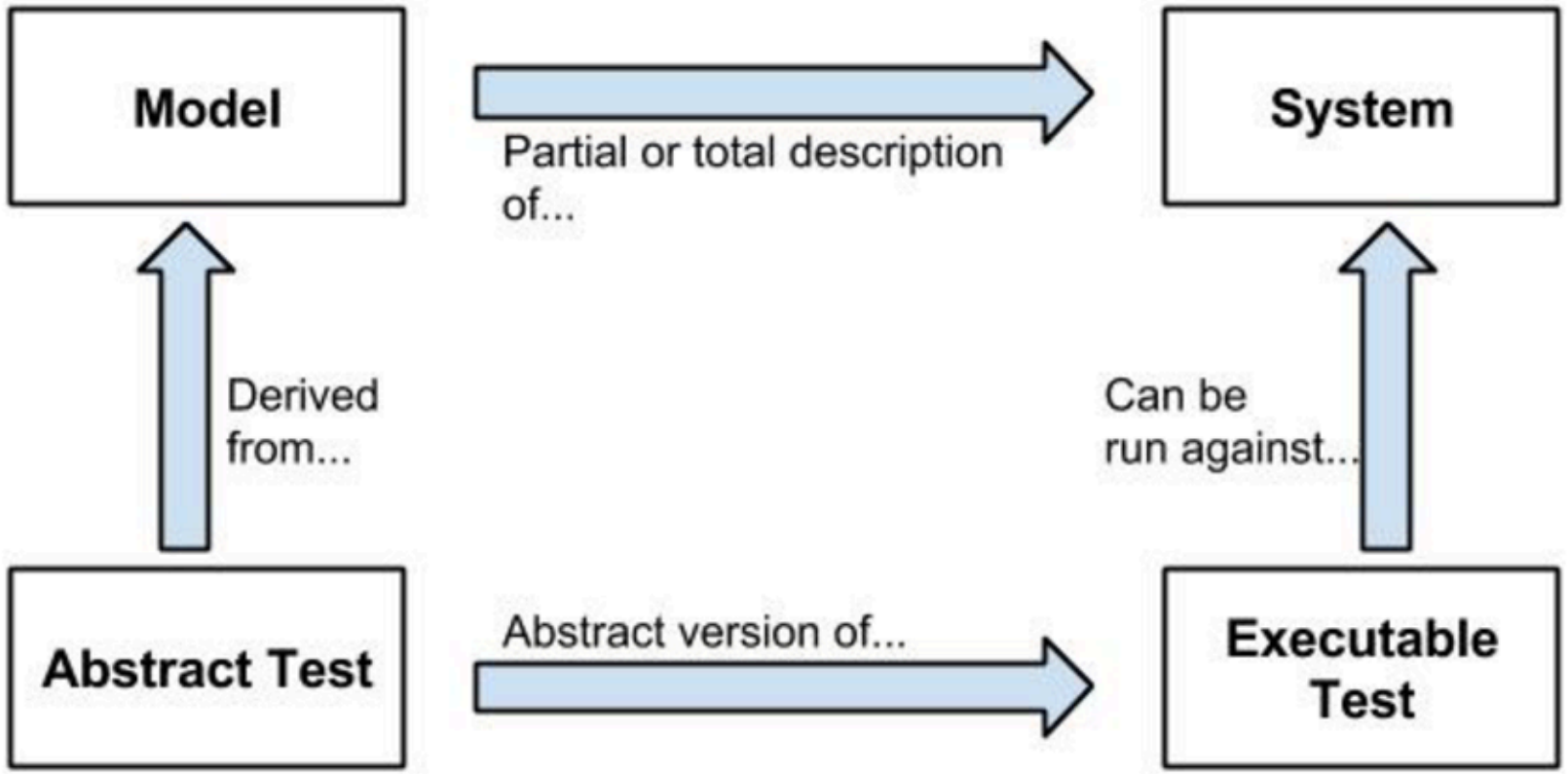
Model-based testing is an application of model-based design for designing and optionally also executing artifacts to perform software testing or system testing. Models can be used to represent the desired behavior of a system under test (SUT), or to represent testing strategies and a test environment.

"All testing is model-based"

- Patterns for test design
  - Methods
  - Classes
  - Package and system integration
  - Regression
  - Test automation
  - Oracles
- In total 35 patterns, each a test meta model



Overview of MBT



From: http://web.imt-atlantique.fr/x-info/atlanmod/index.php?title=Model_Based_Testing

Common testing strategies

- Manual exploratory testing
- Manual testing
- Hard-code testing
- Model-based testing

Advantages of MBT

- Easy to understand and communicate from both the business and developer communities
- Separates (business-) logic from testing code
- Increase the test coverage
- Commercial tools are developed to support model-based testing

Advantages of MBT

- Enable to switch testing tool if needed or support multiple platforms using the same model
- Easier test suite maintenance
- Better test quality
  - no human faults
  - Computer can find more combinations for complex systems than human brain

Drawback of MBT

- Requires a formal specification or model to carry out testing
- Changes to the model might result in a different set of tests
- Test cases are tightly coupled to the model
  - Test quality depends on models!
  - Sometimes, manual checking of tests is required

MBT tools

- Leirios test generator
- MaTeLo
- Qtronic
- Reactis
- Spec Explorer
- TRUST (Simula developed :D)
- ecFeed
- …

Example: Leirios test generator

- Operating system: Windows, Linux
- Modelling:
  - Languages: UML+OCL
  - Supporting third party modelling tools
    - Rational Software Modeler（RSA）
- Model validation
  - Checking generated test cases

Example: Leirios test generator

- Test generation guiding
  - Targets: test are consisting three parts:
    - Pre-
    - Body (=target)
    - Post-
- •Test writing:
  - Exporting test suite
  - Large variety of exporting formats
  - Manual checking
- •Report & traceability
  - Traceability matrix

Choose the tool that fits your need!

Agenda

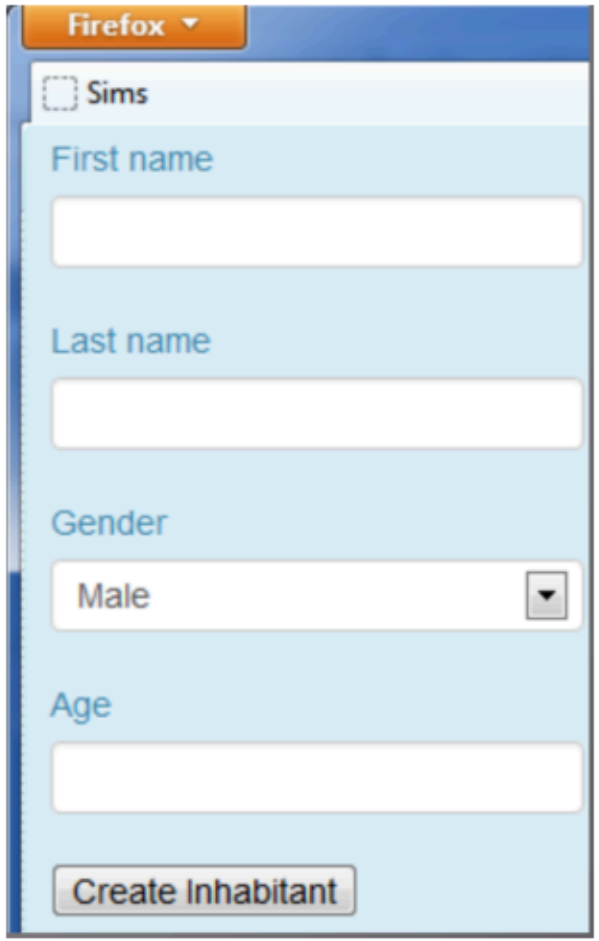
- Introduction (30 min)
- **Start MBT with a simple example (30 min)**
- Short break (10 min)
- The value of models (40 min): three real case studies
- Q&A (10 min)

System under test (SUT)

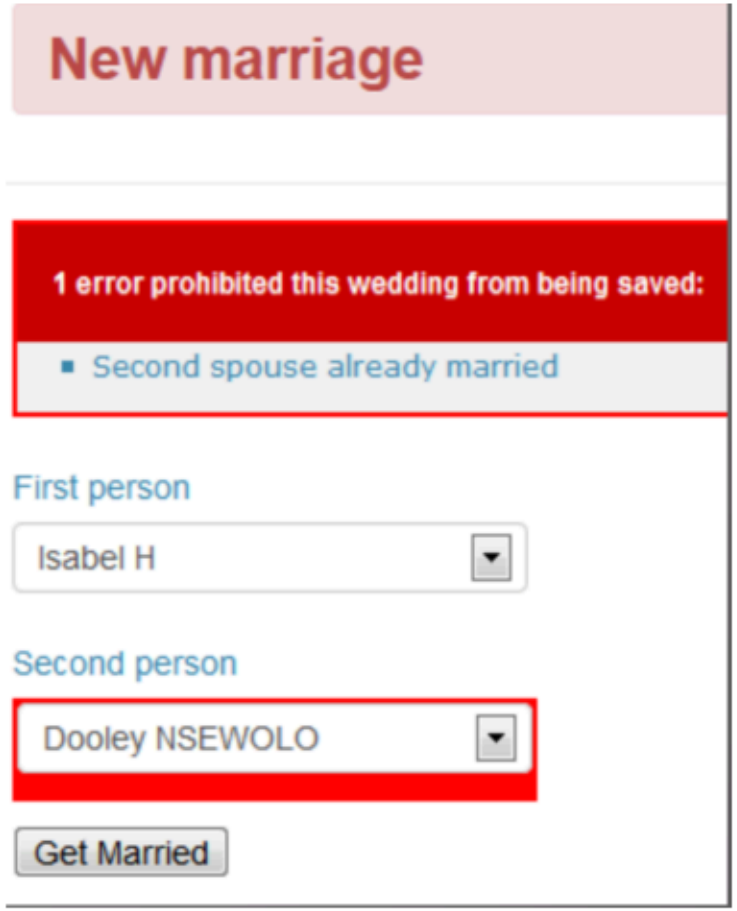
- A simple Web Application*
  - Features
    - Manage the inhabitants (Add / Delete / Edit)
    - Marry the inhabitants
    - Divorce the inhabitants
- Navigation
  - Start at HOME
  - Go to CHURCH to perform a Marriage, once done go back HOME
  - Go to COURT to perform a Divorce, once done: go back HOME

*Example borrowed from: rom Eddie Jaffuel

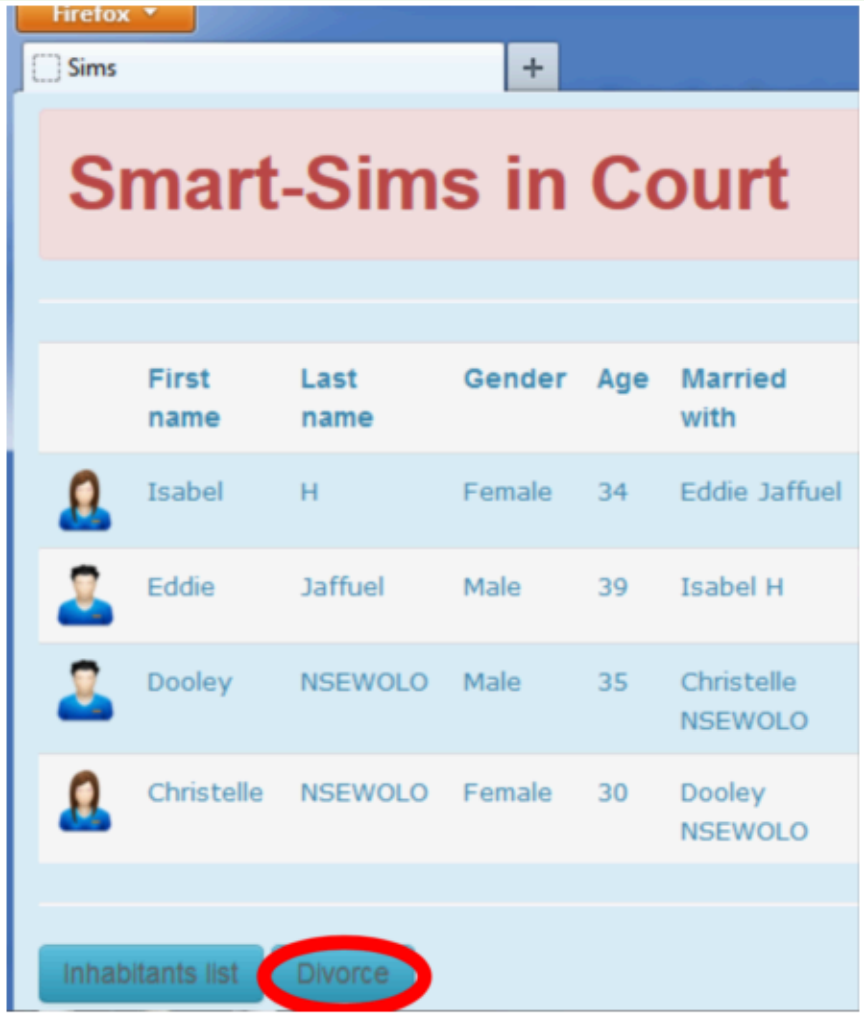
System under test (SUT)



System under test (SUT)



System under test (SUT)



Test requirements (1/2)

@REQ	Requirement description	#AIM			
ADD_INHABITANT	The conditions which allow to add one inhabitant are: - all its informations are provided (identifier, gender, age) - its age have to be greated than 1 - the inhabitant not already exist In case of success, the inhabitant is added to the list of inhabitants, and its information are displayed. A new inhabitant is single by default. If an error occurs, an error message indicates which condition is not fullfilled.	Success	Already exists	Empty identifier	Age not strictly positive
DELETE_INHABITANT	You can only suppress one inhabitant if it exists	Success			

Test requirements (2/2)

@REQ	Requirement description	#AIM				
MARRIAGE	The conditions which allow a marriage are: - age over 18 - none of them are married - one male and one female Once the marriage is accepted, the status of the 2 inhabitants is modified accordingly their fields "Married with ..." is filled If an error occurs, an error message indicates which condition is not fullfilled.	Success	Error same person	Error same gender	Error one is not adult	Error one is already married

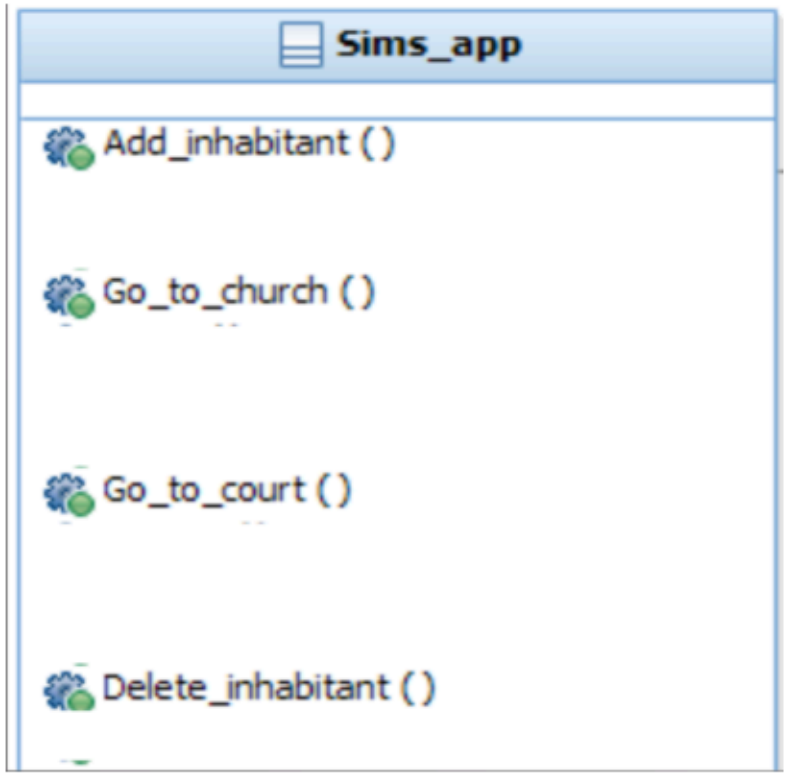
Test model: model system behaviors

- **Behavioral Model**:
  - System functionalities: **Operations**
  - Data representation: **Classes**
  - Initial data of the SUT: **Objects**
  - Behavioral flow: **State Machine**
  - Business rules: **Object Constraint Language (OCL)**

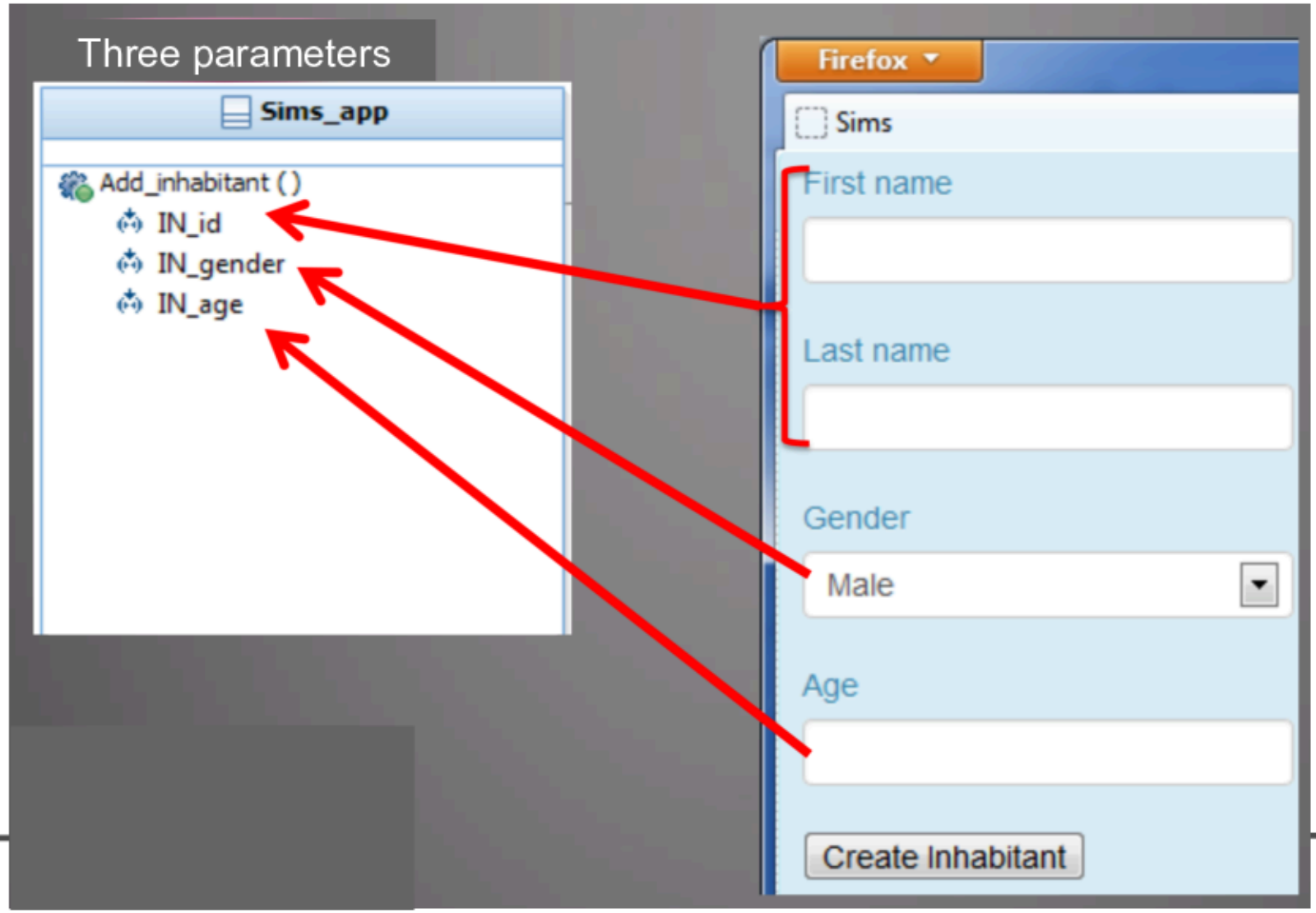
Test model: model system behaviors

- **Behavioral Model**:
  - System functionalities: **Operations**
  - Data representation: **Classes**
  - Initial data of the SUT: **Objects**
  - Behavioral flow: **State Machine**
  - Business rules: **Object Constraint Language (OCL)**

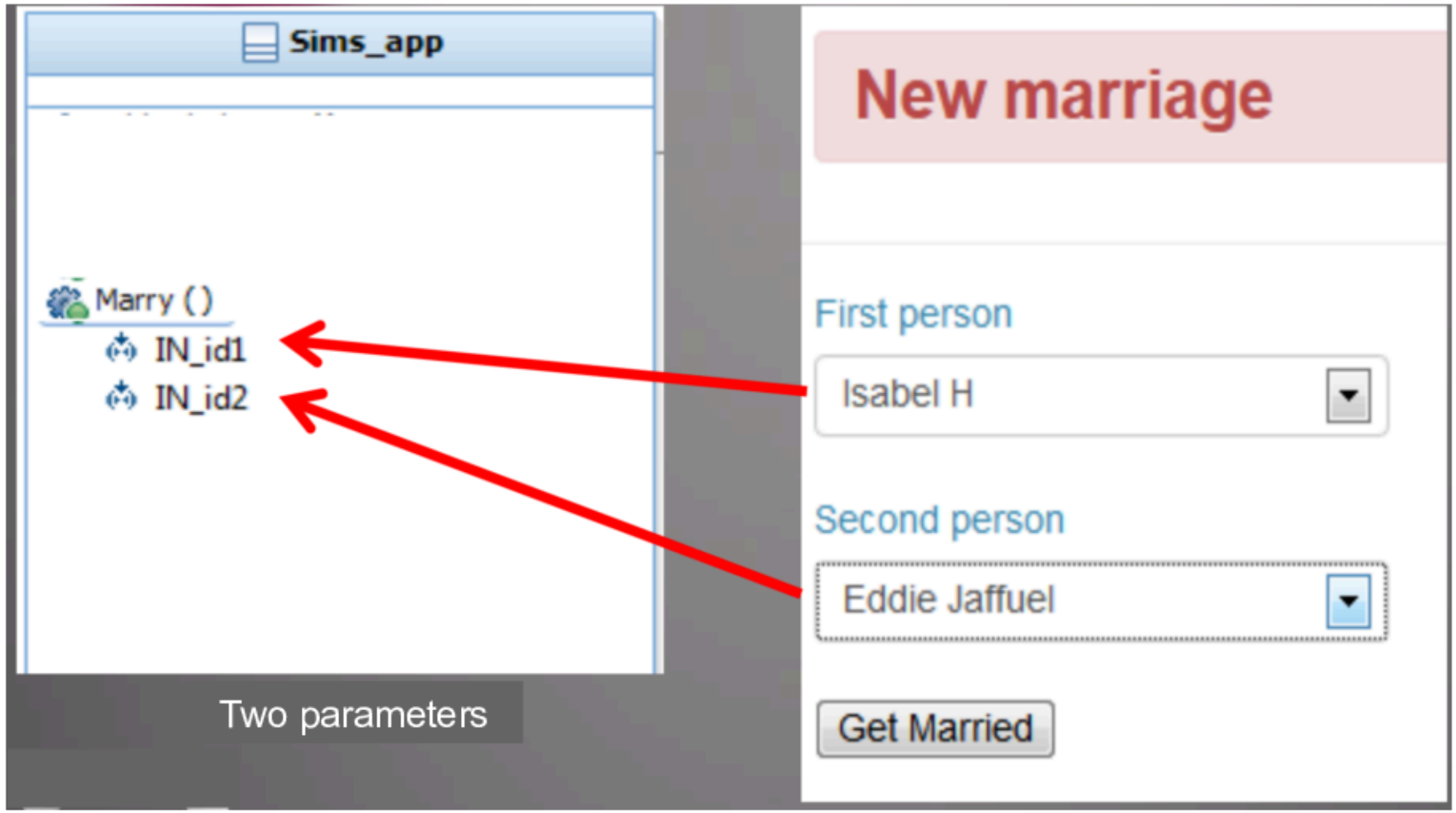
Operations: points of control



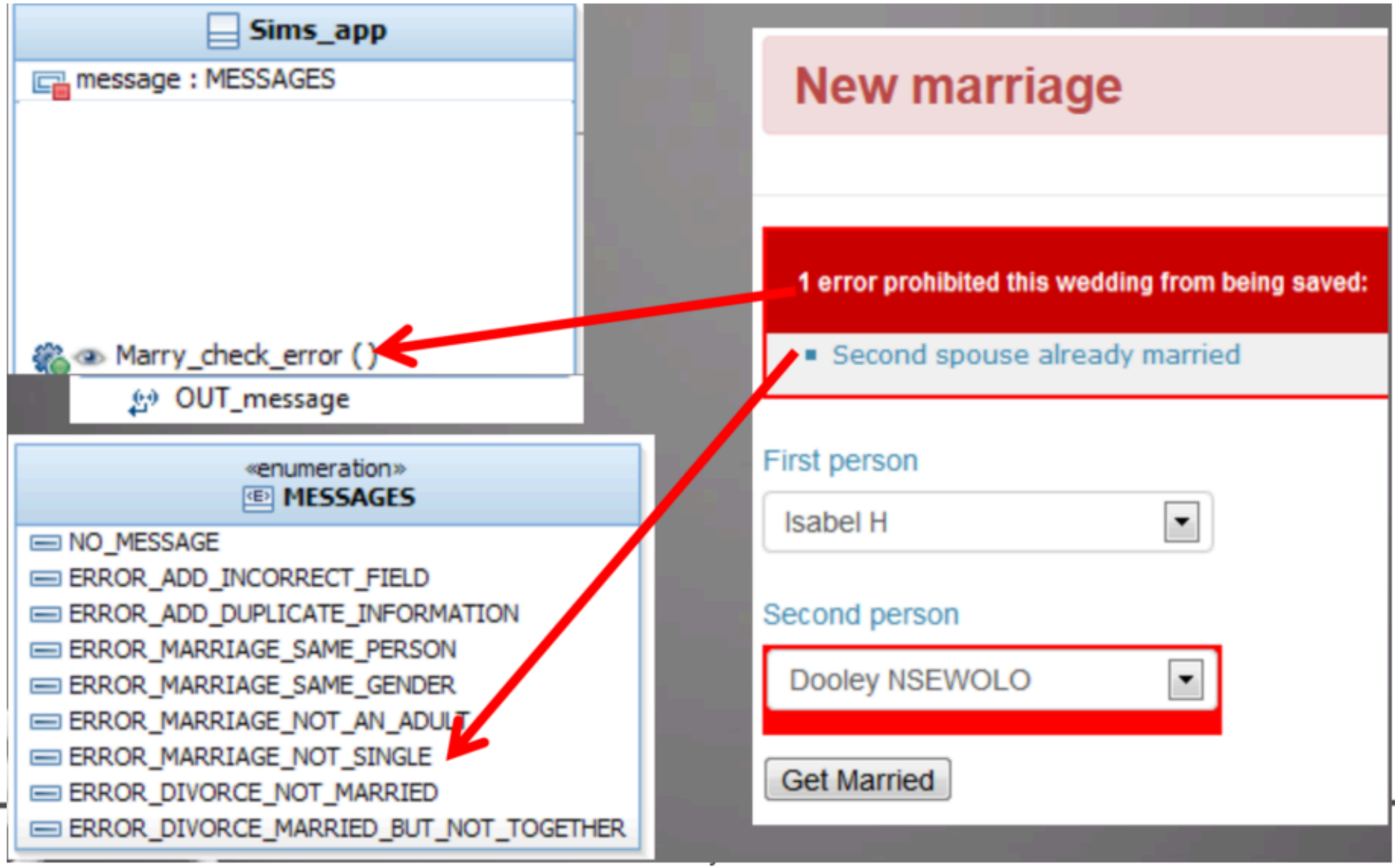
Operations + parameter



Operations + parameter



Operations

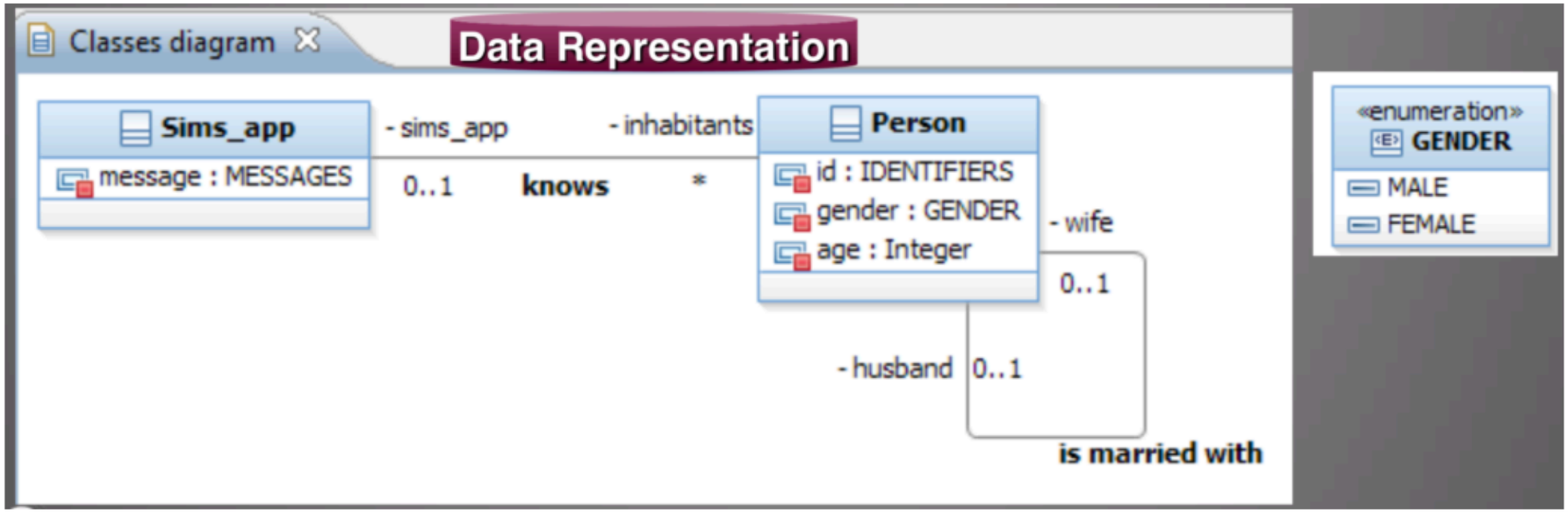

Sims_app
message : MESSAGES
Marry_check_error ( )
OUT_message
«enumeration»
MESSAGES
NO_MESSAGE
ERROR_ADD_INCORRECT_FIELD
ERROR_ADD_DUPLICATE_INFORMATION
ERROR_MARRIAGE_SAME_PERSON
ERROR_MARRIAGE_SAME_GENDER
ERROR_MARRIAGE_NOT_AN_ADULT
ERROR_MARRIAGE_NOT_SINGLE
ERROR_DIVORCE_NOT_MARRIED
ERROR_DIVORCE_MARRIED_BUT_NOT_TOGETHER
New marriage
1 error prohibited this wedding from being saved:
Second spouse already married
First person
Isabel H
Second person
Dooley NSEWOLO
Get Married

Test model: model system behaviors

- **Behavioral Model**:
  - System functionalities: **Operations**
  - Data representation: **Classes**
  - Initial data of the SUT: **Objects**
  - Behavioral flow: **State Machine**
  - Business rules: **Object Constraint Language (OCL)**

Data Representation

- The **Class diagrams** helps to model the Data representation
  - **Classes** helps to represent the Business Entities
  - **Attributes** are the characteristics of the Business Entities
  - **Association** are the relations between Business Entities



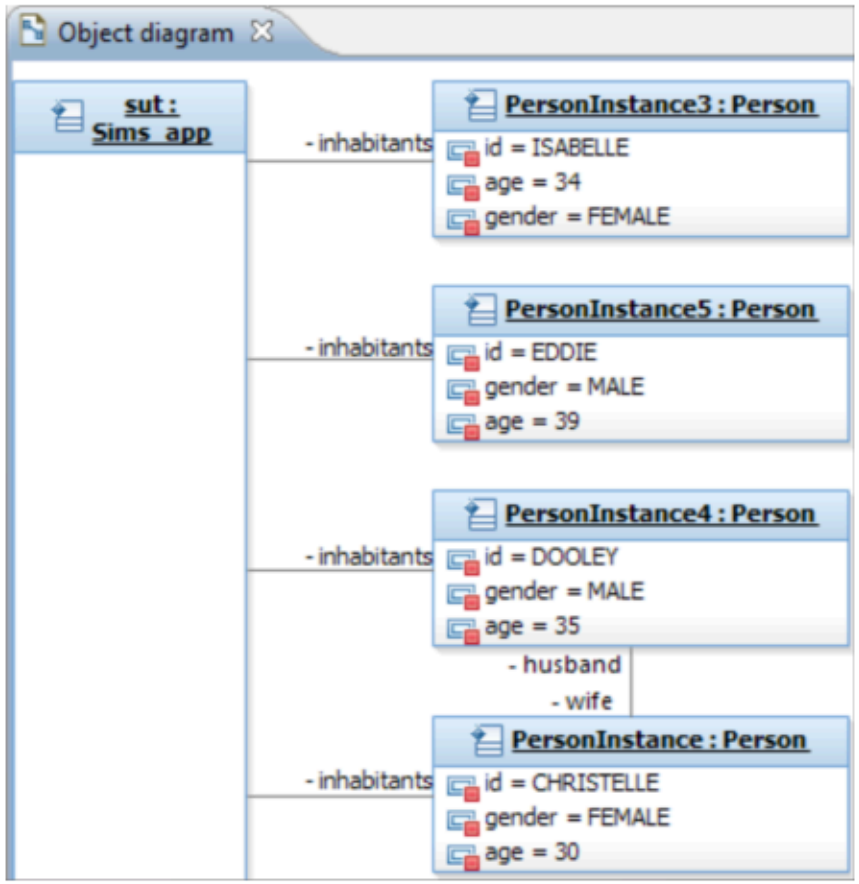
Test model: model system behaviors

- **Behavioral Model**:
  - System functionalities: **Operations**
  - Data representation: **Classes**
  - Initial data of the SUT: **Objects**
  - Behavioral flow: **State Machine**
  - Business rules: **Object Constraint Language (OCL)**

Initial Data of the SUT

- The **Object diagram** used to model the Initial Data of the SUT
  - **Objects** are instance of Business Entities

	First name	Last name	Gender	Age	Married with
	Isabel	H	Female	34	--
	Eddie	Jaffuel	Male	39	--
	Dooley	NSEWOLO	Male	35	Christelle NSEWOLO
	Christelle	NSEWOLO	Female	30	Dooley NSEWOLO

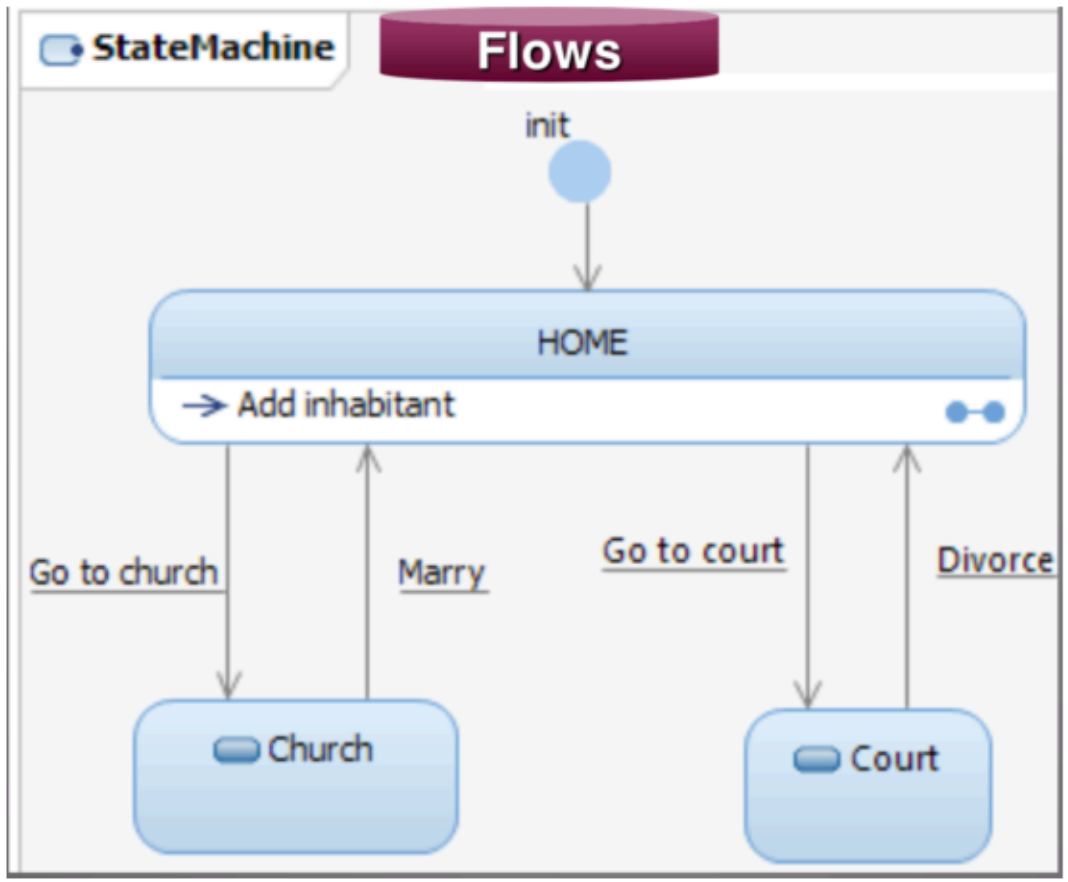


Test model: model system behaviors

- **Behavioral Model**:
  - System functionalities: **Operations**
  - Data representation: **Classes**
  - Initial data of the SUT: **Objects**
  - Behavioral flow: **State Machine**
  - Business rules: **Object Constraint Language (OCL)**

Behavioral flow

- The **State Machine** used to model the Dynamic Flows
  - With **States** and **Transitions**



Test model: model system behaviors

- **Behavioral Model**:
  - System functionalities: **Operations**
  - Data representation: **Classes**
  - Initial data of the SUT: **Objects**
  - Behavioral flow: **State Machine**
  - Business rules: **Object Constraint Language (OCL)**

Business rules: OCL

@REQ	MARRIAGE
Requirement description	The conditions which allow a marriage are: - age over 18 - none of them are married - one male and one female Once the marriage is accepted, the status of the 2 inhabitants is modified accordingly their fields "Married with ..." is filled If an error occurs, an error message indicates which condition is not fullfilled.
#AIM	Error same person
	Error same gender
	Error one is not adult
	Error one is already married
	Success

*Marry

```
---@REQ: MARRIAGE
if person1 = person2 then
    ---@AIM: Error same person
    self.message = MESSAGES::ERROR_MARRIAGE_SAME_PERSON

else if person1.gender = person2.gender then
    ---@AIM: Error same gender
    self.message = MESSAGES::ERROR_MARRIAGE_SAME_GENDER

else if (person1.age < 18) or (person2.age < 18) then
    ---@AIM: Error one is not adult
    self.message = MESSAGES::ERROR_MARRIAGE_NOT_AN_ADULT

else if person1.is_married() or person2.is_married() then
    ---@AIM: Error one is already married
    self.message = MESSAGES::ERROR_MARRIAGE_NOT_SINGLE

else
    ---@AIM: Success
    self.message = MESSAGES::NO_MESSAGE
    and
    if person1.gender = GENDER::MALE then
        ---@AIM: person1 is a man
        person1.wife = person2
    else
        ---@AIM: person1 is a woman
        person1.husband = person2
    endif
```

Test generation

Firefox

Requirement	Aims	Tests
MARRIAGE *The conditions which allow a marriage are:* *- age over 18* *- none of them are married* *- one male and one female* *Once the marriage is accepted, the status of the 2 inhabitants is modified accordingly their fields "Married with ..." is filled* *If an error occurs, an error message indicates which condition is not fullfilled.*	Error one is already married	testSuite_Family Marry_Error_one_is_already_married (02-23-c8)
	Error one is not adult	
	Error same gender	testSuite_Family Marry_Error_same_gender (02-9a-ed)
	Error same person	testSuite_Family Marry_Error_same_person (02-2a-e4)
	Success	testSuite_Family Marry_Success (02-ae-f1)
DIVORCE *The conditions which allow a divorce are:* *- 2 person being married together* *Once the divorce is accepted, the status of the 2 inhabitants is modified accordingly: their fields "Married with ..." is empty.*	Error married but not together	testSuite_Family Divorce_Error_married_but_not_together (02-24-87)
	Error one is not married	testSuite_Family Divorce_Error_one_is_not_married (02-91-7c)
	Success	testSuite_Family Divorce_Success (02-57-2b)

Test example: 1

Firefox

Test: Marry_Error_same_person (02-2a-e4)

Steps	Actions	Requirements, aims
Step 1 *(sut)*	*Go to church* Click on the button "Go to church"	
Step 2 *(sut)*	*Marry* Click on "Get Married" Select the first person identified with EDDIE Select the second person identified with EDDIE Click on the bouton "OK"	**REQ** MARRIAGE **AIM** Error same person
2.1	*Check that the message ERROR_MARRIAGE_SAME_PERSON is displayed*	

Test example: 2

Firefox

Test: Marry_Error_one_is_already_married (02-23-c8)

Steps	Actions	Requirements, aims
Step 1 *(sut)*	*Go to church* Click on the button "Go to church"	
Step 2 *(sut)*	*Marry* Click on "Get Married" Select the first person identified with DOOLEY Select the second person identified with CHRISTELLE Click on the bouton "OK"	**REQ** MARRIAGE **AIM** Error one is already married
2.1	*Check that the message ERROR_MARRIAGE_NOT_SINGLE is displayed*	

Take-away from this example…

- **Step 1: build test model**
  - **Operations:** points of control and observation
  - **Classes**: data representation
  - **Object:** initial data of the SUT
  - **State** Machine: business flows
  - **Object Constraint Language (OCL):** business rules
- **Step 2: generate tests**

Agenda

- Introduction (30 min)
- Start MBT with a simple example (30 min)
- **Short break (10 min)**
- The value of models (40 min): three real case studies
- Q&A (10 min)

Agenda

- Introduction (30 min)
- Start MBT with a simple example (30 min)
- Short break (10 min)
- **The value of models (40 min): three real case studies**
- Q&A (10 min)

Three case studies using models

- Cisco: Video Conferencing System (VCS)
  - Case study 1: generating tests for testing robustness of VCS
  - Case study 2: test selection for VCS regression testing
    - Not using UML and OCL!
    - Broadly, models in MBT can be any models!
- Kreftregisteret: Cancer Registry System (CRS)
  - Case study 3: model-based framework for supporting automated CRS
    - Not for testing but interesting.

Case study 1

MBT for Video Conferencing Systems (VCSs)

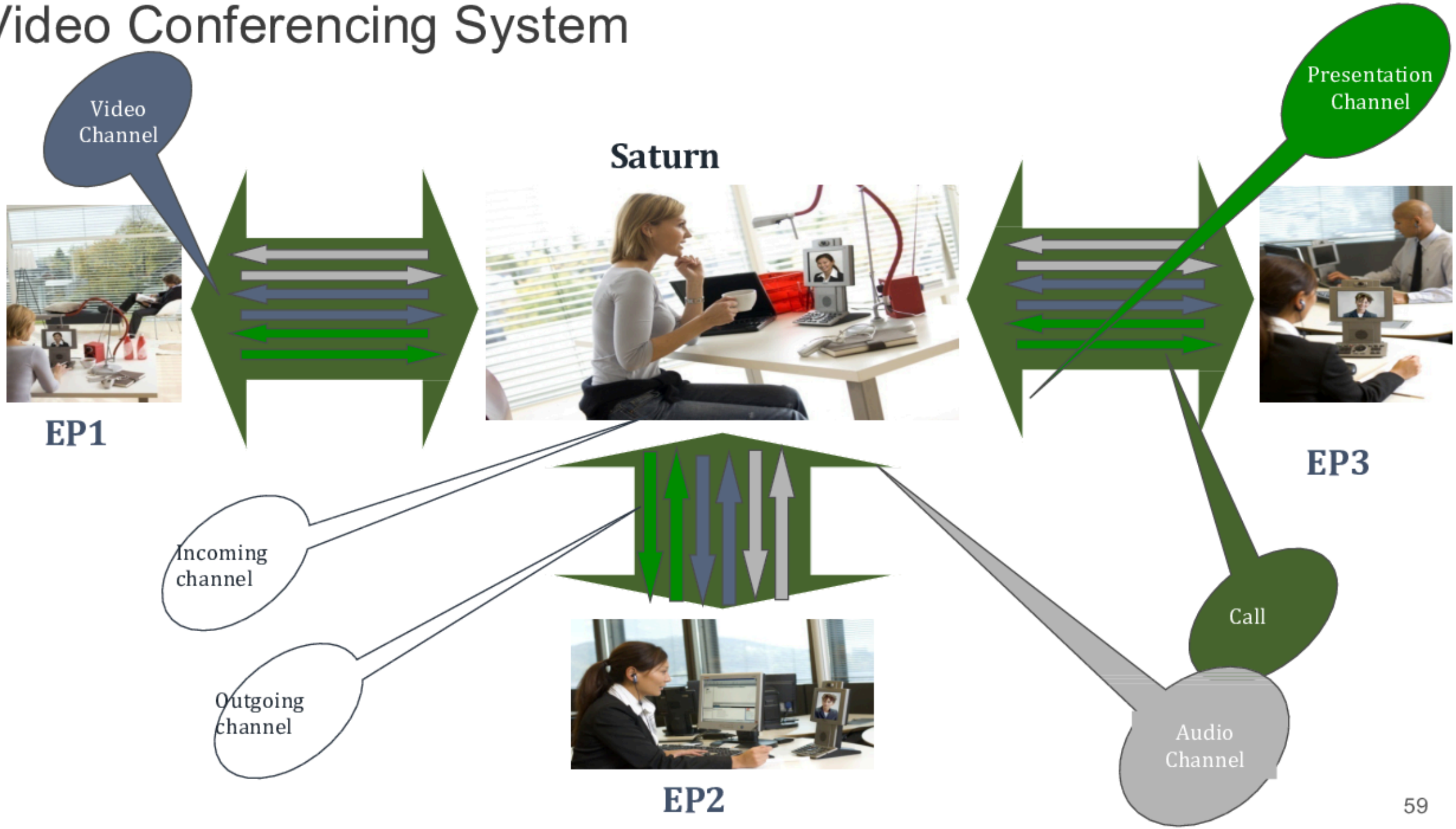
C20

C40

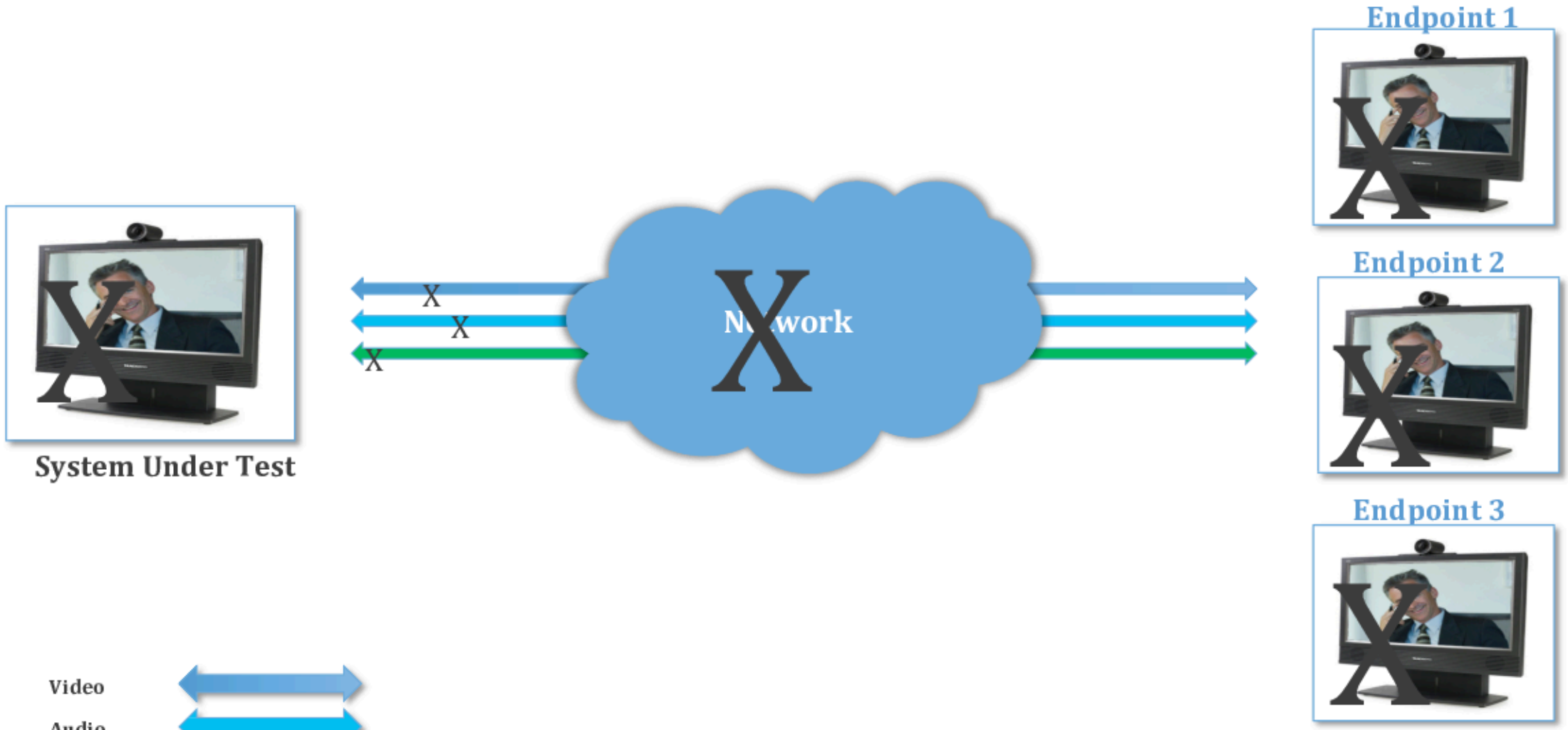
C60

C90

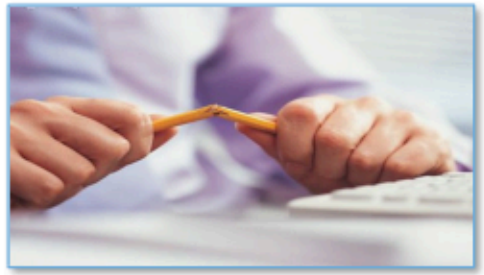
Video Conferencing System



Model-based Robustness Testing



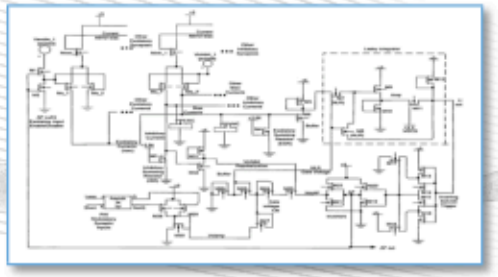
Modeling Robustness Behavior



❖ **Robustness is the degree to which a software component functions correctly in the presence of exceptional inputs or stressful environmental conditions (IEEE Std 610.12-1990)**

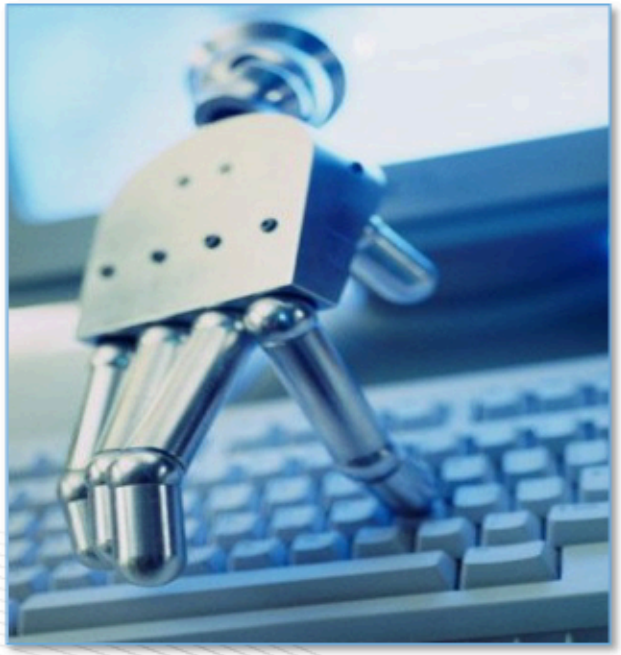
❖ **Modeling robustness behavior of Cisco's Video Conferencing System (VCS) in the presence of faults**

- ❖ Network communication faults
- ❖ Media quality faults in media streams
- ❖ Faults in the endpoints



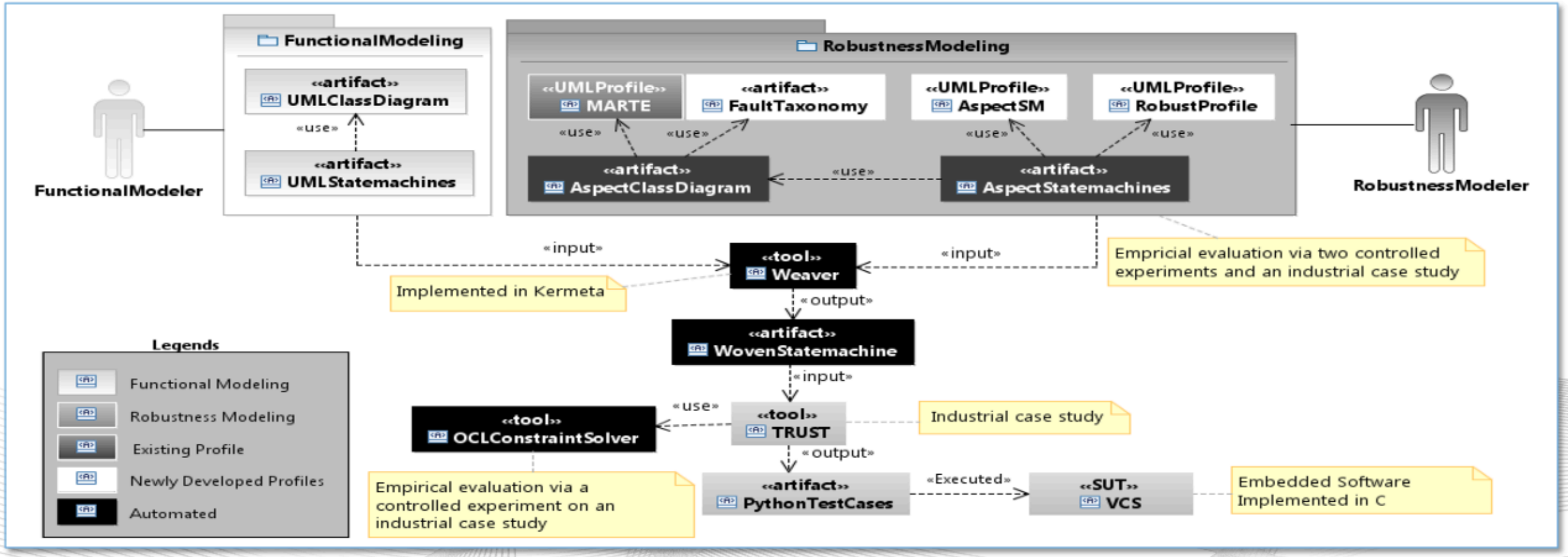
❖ **Modeling robustness behavior, for example in state machines, makes modeling highly complex and redundant**

Testing Robustness Behavior



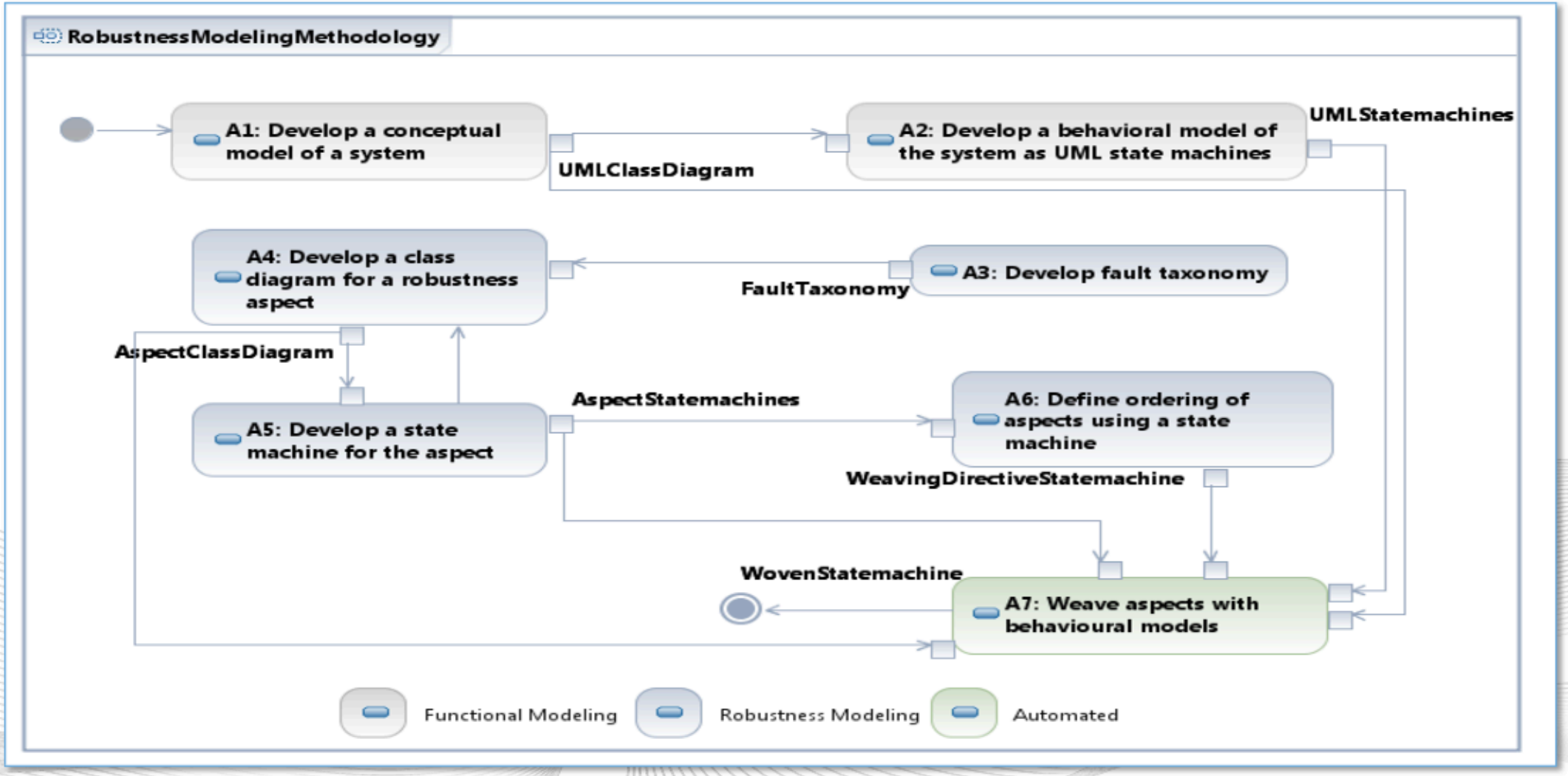
- **Automated generation of executable test cases from robustness models**
  - Specifically targeting to identify robustness faults that can occur in a system due to faulty situations in the environment of the system
  - Automation of this process requires many steps:
    - Defining appropriate test strategies aimed to perform robustness testing
    - Using search-based techniques to generate test data from OCL constraints

Solution for Model-based Robustness Testing



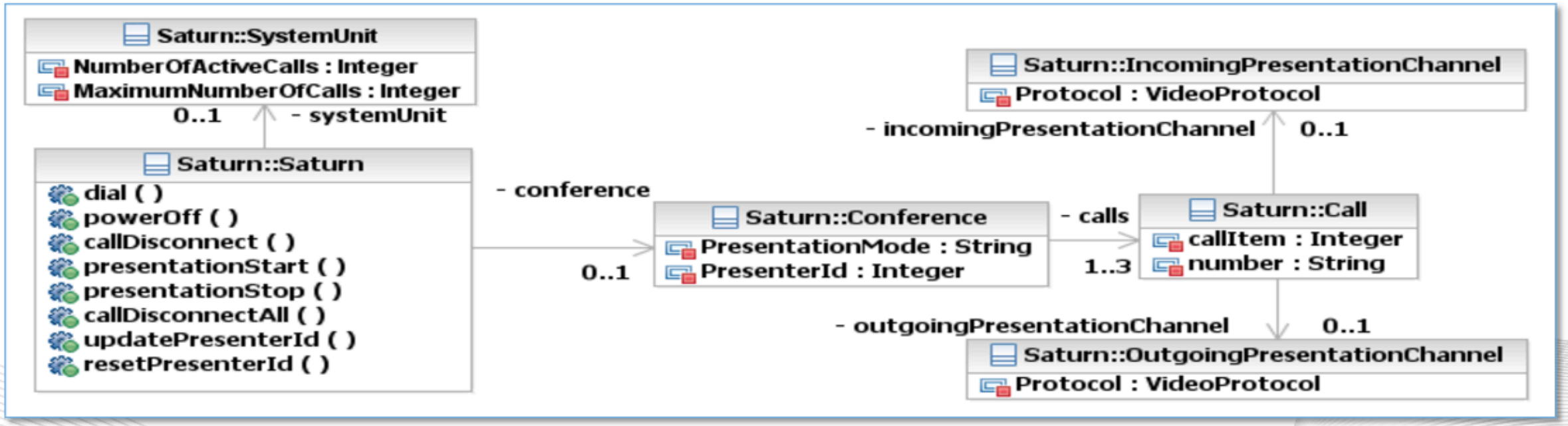
Modeling Robustness Behavior

Robustness Modeling Methodology (RUMM)

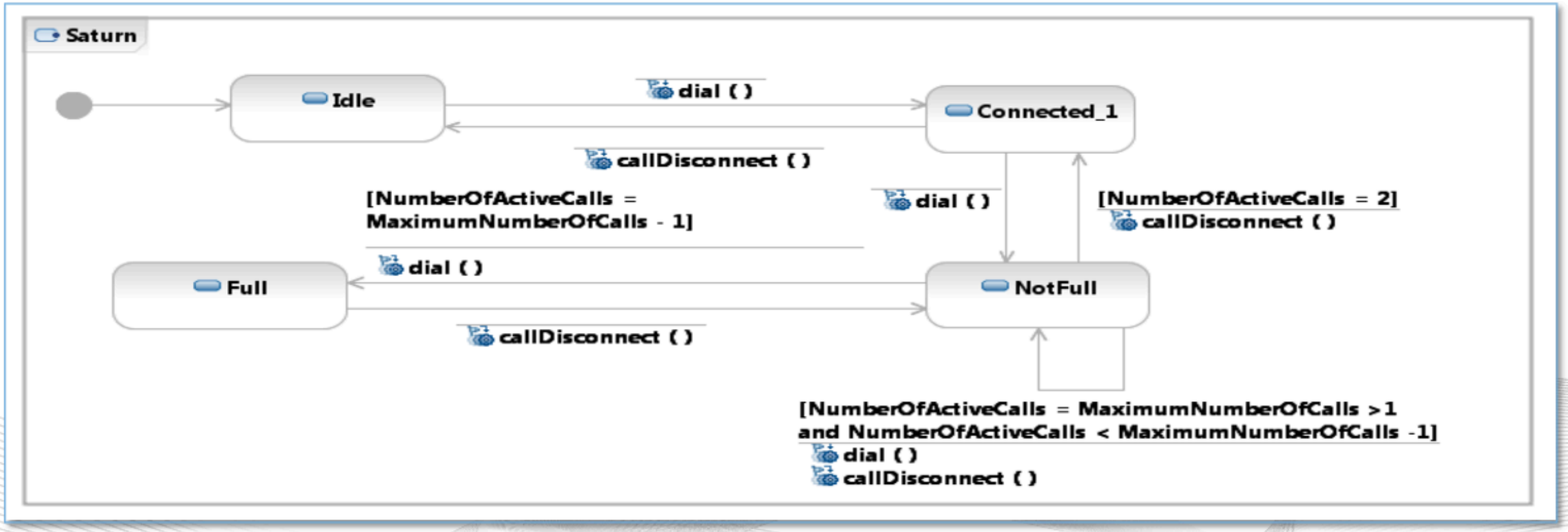


* S. Ali, L. Briand, and H. Hemmati. Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems, Accepted for publication in the Journal of Software and Systems Modeling, Springer, 2011

Activity A1: Class Diagram of the System



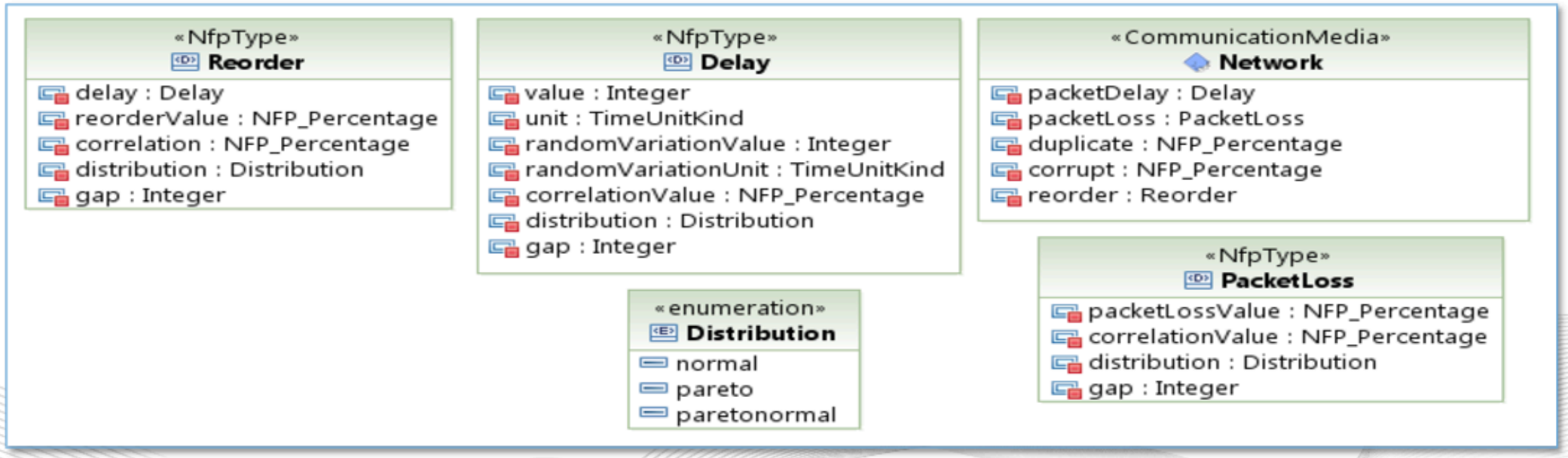
Activity A2: Develop Behavioral Model



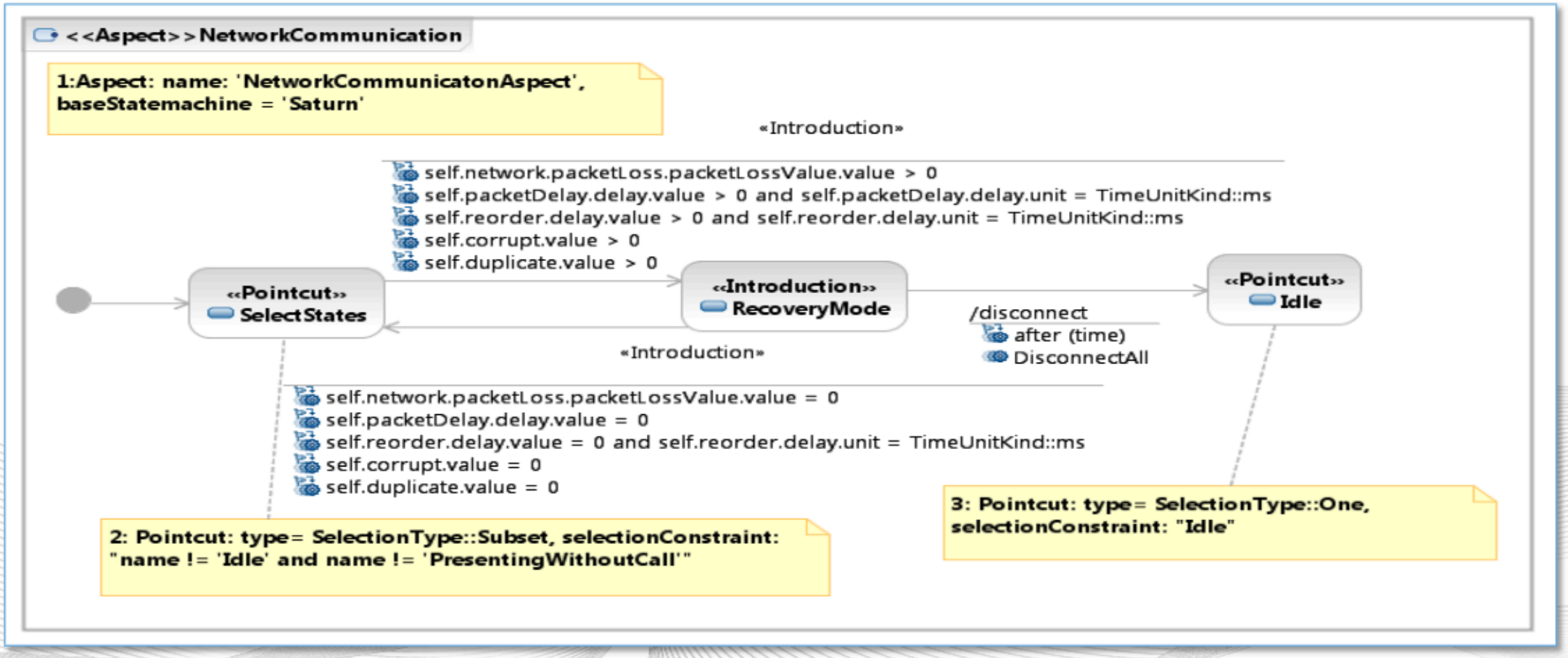
Activity A3: Develop Fault Taxonomy

Fault	Description of the fault
Packet Loss	This fault introduces network packet loss during a videoconference
Jitter	This fault introduces delays in the packet during a videoconference
Illegal H323 packet	This fault introduces illegal/malformed H323 packets in a H323 videoconference
Illegal SIP packet	This fault introduces illegal/malformed SIP packets in a SIP videoconference
No network connection	This fault shut downs the network
Low bandwidth	This fault reduces the bandwidth of the network to less than the bandwidth required by a videoconference

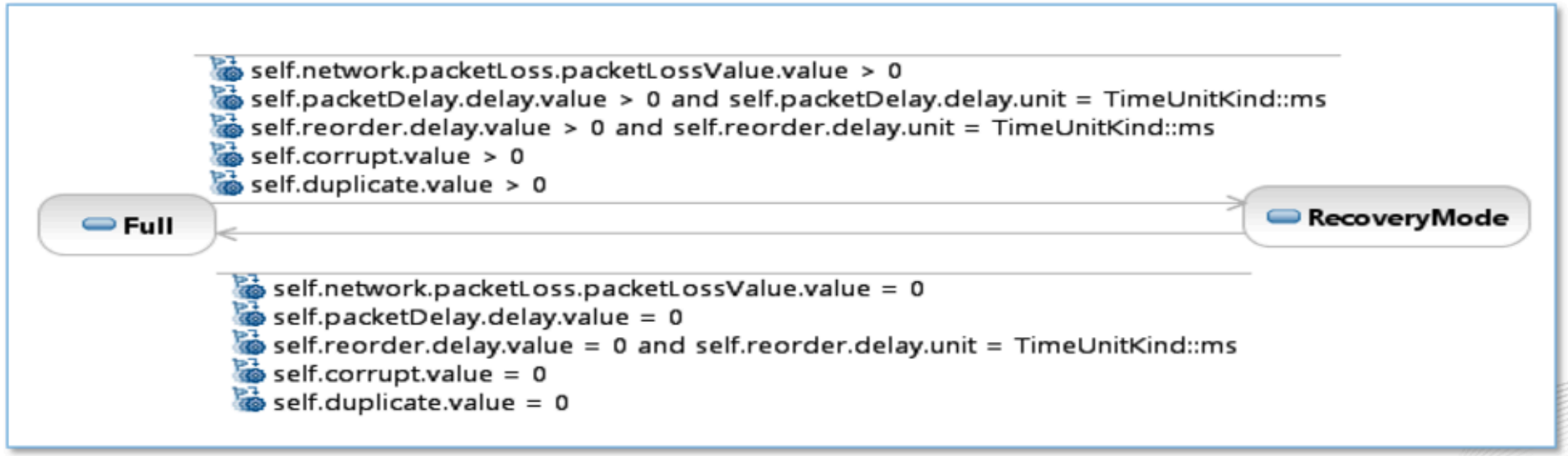
Activity A4: Develop Class Diagram for Robustness Aspect



Activity A5: Develop Aspect State Machine



Activity A7: Weave Aspect State Machines



Evaluation- Reduced Modeling Effort

Crosscutting behavior	Using aspects			Without aspects			Effort Saved (%)		
	States (Added)	Transition (Added)	Trigger (Added)	States (Modified/ Added)	Transitions (Modified/Added)	Trigger (Added)	States	Transitions	Trigger
Updating audio constraints	1	-	-	86 (Modified)	-	-	**98%**	-	-
Updating video constraints	1	-		86 (Modified)	-	-	**98%**	-	-
Media quality recovery	3	3	19	20 (Added)	178	1604	-	**98%**	**98%**
Network communication	3	3	13	20 (Added)	178	1082	-	**98%**	**98%**
Add Guard	2	1	-	0	22 (Modified)	-	-	**95%**	-

* S. Ali, L. Briand, and H. Hemmati. Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems, Accepted for publication in the Journal of Software and Systems Modeling, Springer, 2011

Experiences -- Robustness Modeling

- Approximately 98% of modeling effort saved
- Improved readability of models evaluated based on two controlled experiments [1]
  - Readability (measured as defect identification and correction) was significantly better than standard UML state machines. On average 28% increased.
  - No significant differences in readability measured in terms of comprehension questionnaire
  - No significant difference in effort was observed
- Easier model evolution
- Enhanced separation of concerns

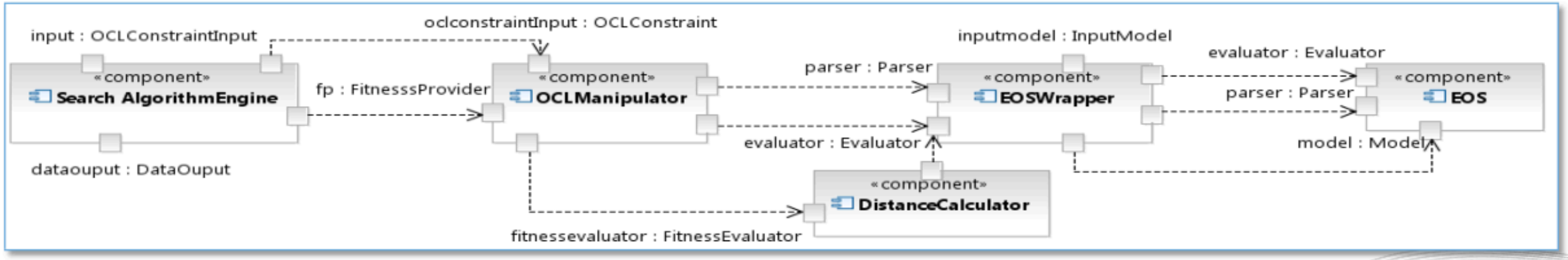
1. S. Ali, T. Yue, and L. Briand. Does Aspect-Oriented Modeling Help Improve the Readability of UML State Machines?.

Test Case Generation

Test Case Generation

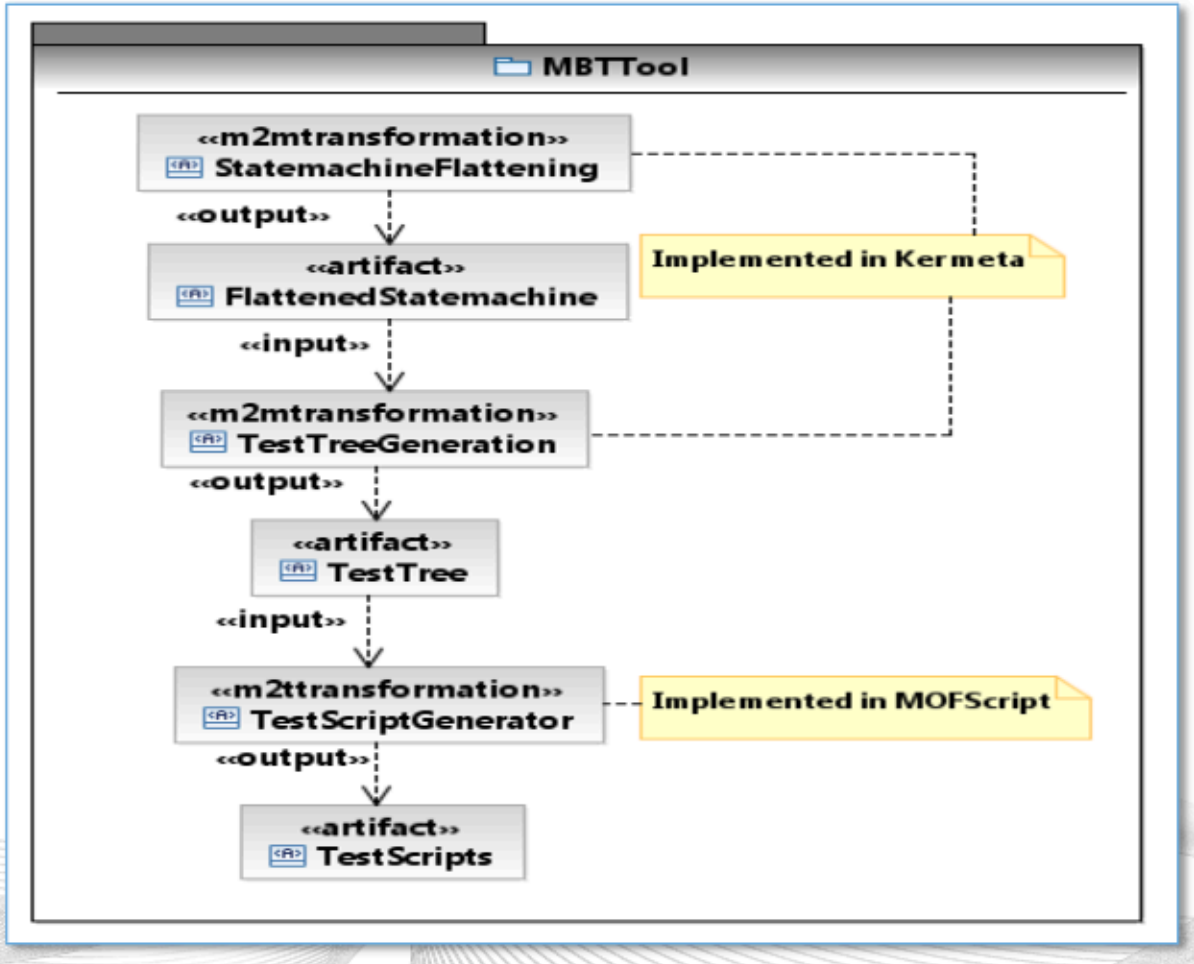
- **Constraints solving using search algorithms**
  - Developed a tool to generate test data required to violate different properties of the environment to check robustness of the system against those situations.
  - Used evolutionary algorithms such as GA, 1+1 (EA), and AVM and defined branch distance function for operations specific to OCL
- **Developed a tool TRansformation-based tool for Uml-baSed Testing (TRUST)**
  - Supports configurable and extensible features such as input models, test models, coverage criteria, test data generation strategies, and test script languages.

Architecture Diagram for Search-based Solver for OCL



1. EyeOCL (EOS) Software, http://maude.sip.ucm.es/eos/, 2010

The TRUST Tool



S. Ali, H. Hemmati, N. E. Holt, E. Arisholm, and L. Briand. Model Transformations as a Strategy to Automate Model-Based Testing - A Tool and Industrial Case Studies, Simula Research Laboratory, Technical Report(2010-01), 2010.

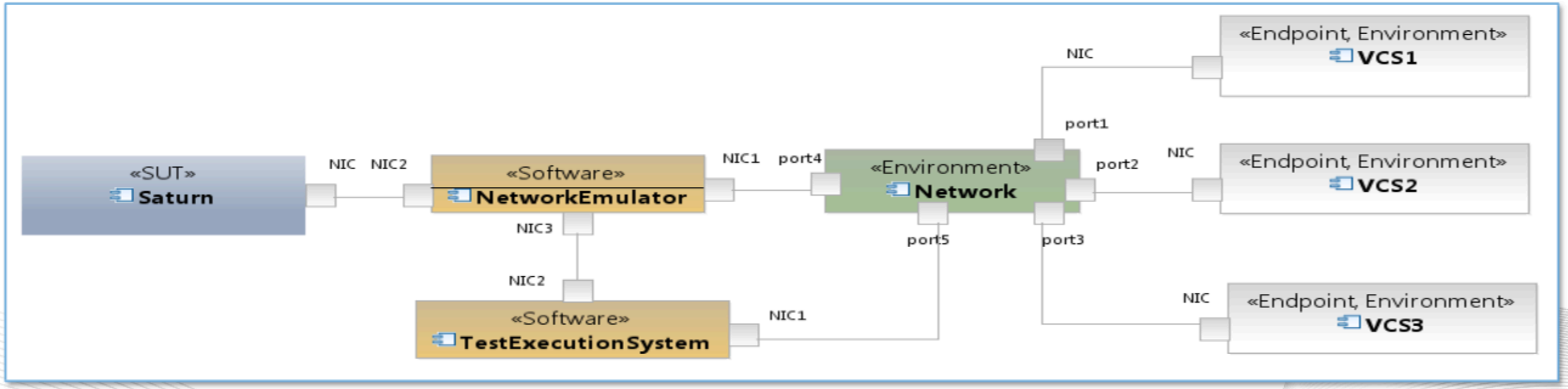
Experiences – Test Case Generation

- **A few approaches are developed for OCL constraint solving, but they have one or more of the following problems**
  - Target only a small subset of OCL
  - Not scalable due to translations in to a formalism such as Satisfiability Problem (SAT) and Constraint satisfaction problem (CSP)
  - Lack of proper tool support
- **Results of our solver for 57 industrial constraints:[1,2]**
  - AVM (100%), (1+1) EA (98%), GA (65%), RS (49%)
  - UML2CSP could not solve constraints ranging from 5-8 clauses (47 constraints). We ran each constraint for one hour.
- **The TRUST tool has been used for two industrial case studies (Cisco and ABB)[3]**

1. S. Ali, M. Z. Iqbal, A. Arcuri, and L. Briand. A Search-based OCL Constraint Solver for Model-based Test Data Generation, In: Proceedings of the 11th International Conference On Quality Software (QSIC 2011), pp. 41-50, IEEE, 2011.
2. S. Ali, M. Z. Iqbal, A. Arcuri, and L. Briand. Solving OCL Constraints for Test Data Generation in Industrial Systems with Search Techniques , Submitted to ACM Transactions on Software Engineering and Methodology (TOSEM), 2011.
3. S. Ali, H. Hemmati, N. E. Holt, E. Arisholm, and L. Briand. Model Transformations as a Strategy to Automate Model-Based Testing - A Tool and Industrial Case Studies, Simula Research Laboratory, Technical Report(2010-01), 2010.

Test Case Execution

Test Case Execution



Experiences – Test Case Execution

- Execution of test cases found one critical fault, when tests were executed on a small subsystem

- Executing robustness test cases is expensive because it requires setting up special equipment (hardware and/or software-based emulators)

- Running one robustness test case requires booking a specialized testing lab and takes on average 15 minutes on a Cisco's VCS

Case study 2

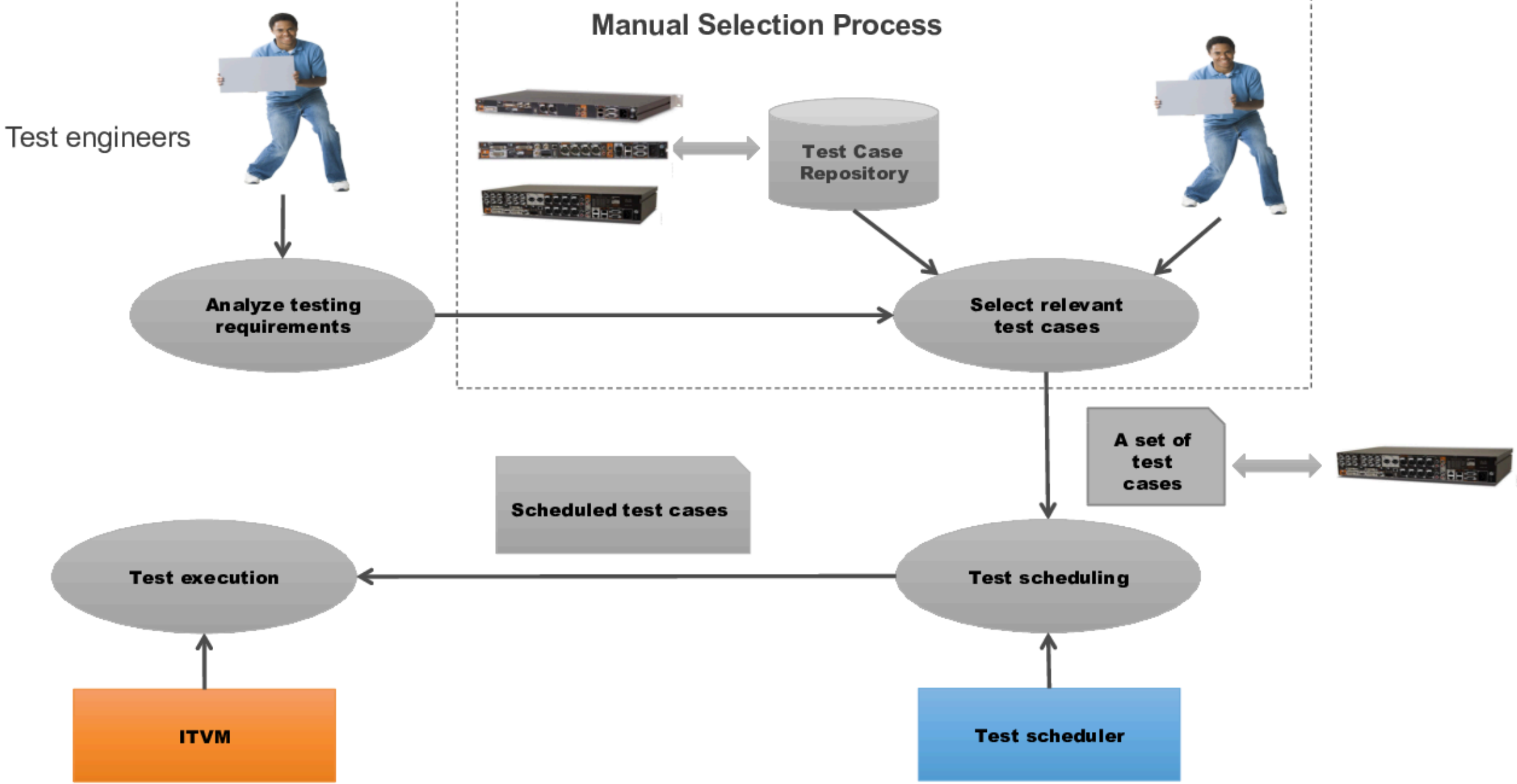
Test case selection using feature model

C20

C40

C60

C90


Manual Selection Process
Test engineers
Test Case Repository
Analyze testing requirements
Select relevant test cases
A set of test cases
Scheduled test cases
Test execution
Test scheduling
ITVM
Test scheduler

Challenges for Manual Test Case Selection

- Time-consuming for manual selection
- Driven by the expertise, not objective and repeatable
- Low test coverage for the selected test cases
- No systematic guidelines for new test engineers

Overall View

- Feature Model to capture the commonalities and variabilities of product line
- Component Family Model for Testing (CFM_T) to model the structure of a large number of test cases in the repository

Feature Model for Testing (FM_T)

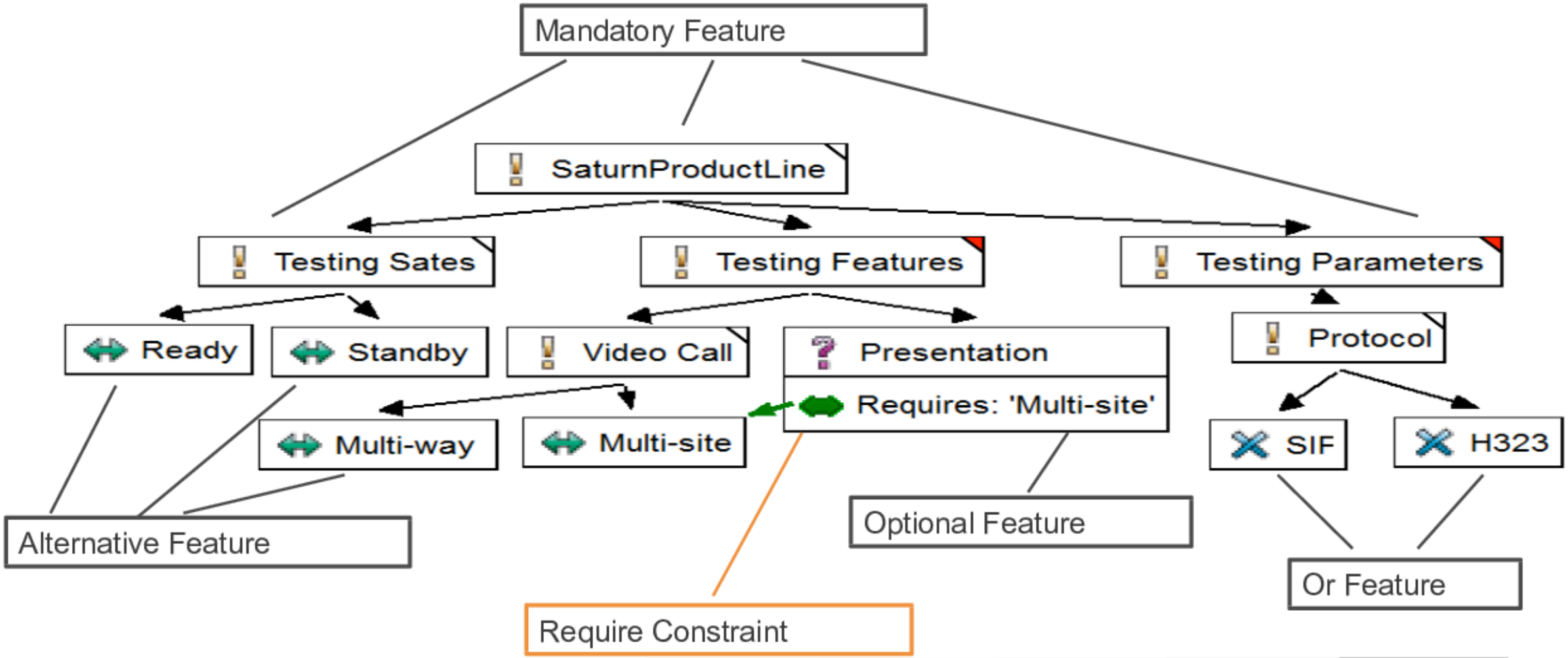
- Feature modeling is a hierarchical modeling approach for capturing commonalities and variabilities in PL

- VCS PL is composed of distinct products that can be configured in different ways. We designed the VCS PL using Feature Model for Testing (FM_T)

FM_T

- FM can be represented as ***FM = {features, constraints}***
- ***Features= {mandatory, optional, alternative, or}***
- ***Constraints = {require, mutually exclusive}***

FM_T


Mandatory Feature
SaturnProductLine
Testing Sates
Testing Features
Testing Parameters
Ready
Standby
Video Call
Presentation
Requires: 'Multi-site'
Protocol
Multi-way
Multi-site
SIF
H323
Alternative Feature
Optional Feature
Or Feature
Require Constraint

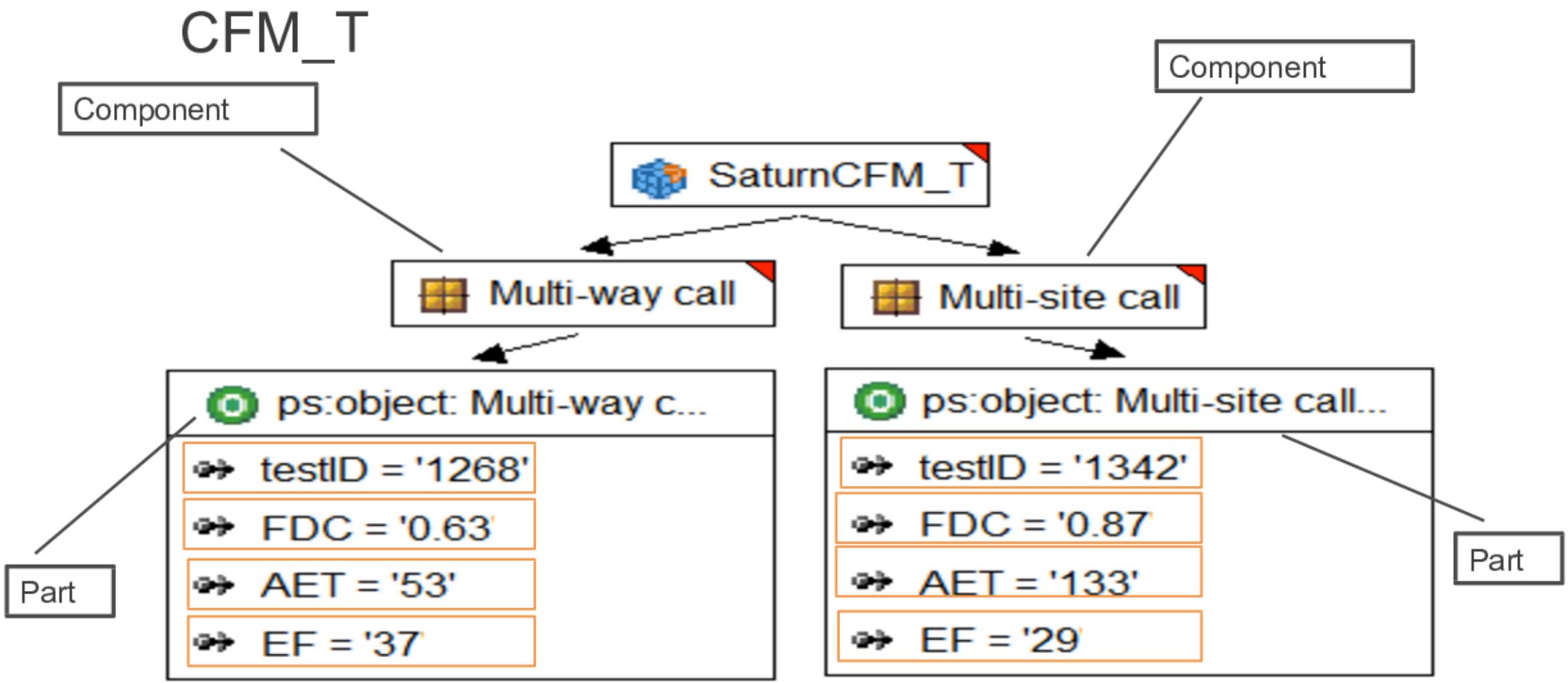
Component Family Model for Testing (CFM_T)

- Usually, CFM is used to model the software system architecture in product line, thereby facilitating product configuration.

- In our approach, CFM is not used to model the software architecture, but rather to model a large number of test cases in the repository.

CFM_T

- CFM_T can be represented as ***CFM_T = {component, part, restrictions}***
- A ***component*** represents a test suite or a test task that can be regarded as a named set of test cases
- A ***part*** represents a test case that belongs to a test task
- *A* ***restriction*** specifies whether an element in CFM_T can be apart of the final decision

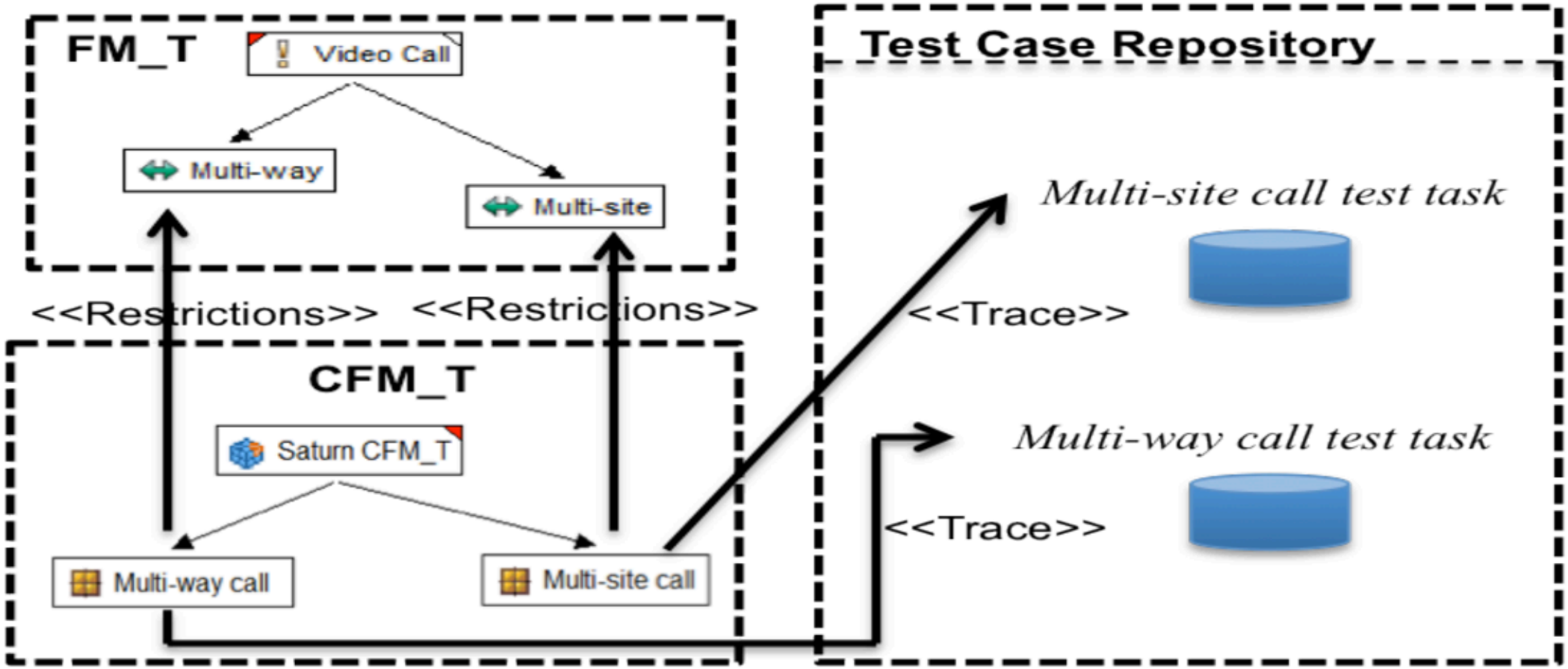

CFM_T
Component
Component
SaturnCFM_T
Multi-way call
Multi-site call
ps:object: Multi-way c...
testID = '1268'
FDC = '0.63'
AET = '53'
EF = '37'
ps:object: Multi-site call...
testID = '1342'
FDC = '0.87'
AET = '133'
EF = '29'
Part
Part

Summary for FM_T and CFM_T

FM_T	Features	Mandatory	44
		Alternative	38
		Optional	25
		Or	27
	Constraints	Require	35
		Exclusive	0
CFM_T	Components		143
	Parts		2374
	Attributes		9496
	Restrictions		7386

- Building FM_T is one-time manual effort
- CFM_T can be built from existing tese case information

Summary for FM_T and CFM_T



Discussion

- ❑ Abstraction and automation
  - Hide the implementation details
  - No need to go through test cases manually
- ❑ Less reliance on domain expertise
- ❑ Reduced maintenance effort
- ❑ Adaption in other context

Questionnaire-Based Survey

- Objective
  - Solicit opinions from the industrial people about their experience for the FM_T and CFM_T

- Plan and design
  - Either multiple choices or required responses on a five-point Likert Scale
  - Four experienced people involved from the current testing team

- Results show test engineers are positive about adapting our methodology to the current practice

Conclusion

- A product line modeling methodology for automated test case selection
  - Feature Model for Testing (FM_T) to model testing functionalities of a product line
  - Component Family Model for Testing (CFM_T) to model the structure of a large number of test cases in the repository
- Evaluation using an industrial case study and questionnaire-based survey
- Selection effort reduced significantly and positive attitudes from test engineers

Case study 3

- **A Model-Based Framework for Supporting An Automated Cancer Registry System**

Context

- Playing cancer research or releasing national report requires sufficient input for cancer
  - Diagnosis
  - Treatment
  - Relapse
  - Death
  - …

CANCER Registry of Norway
INSTITUTE OF POPULATION-BASED CANCER RESEARCH

Context-Cancer registry of Norway (CRN)

Transfer from a paper-based/manual registry system to an ICT-based Automated Cancer Registry System (ACRS)

Cancer messages and cases

- A **cancer message** includes a set of fields (e.g., message type) records all the necessary information related with one cancer about one cancer patient from one specific medical entity, e.g., pathology laboratory.

- A **cancer case** consists of a number of fields (e.g., cancer type) for one cancer about one cancer patient that are necessary for cancer research and report.
  - Aggregated from a set of corresponding cancer messages

Three Key activities in CRN

- **Cancer Message Validation (Step 1)**
  - Collect cancer messages from different medical entities, and check their validity and correctness
    - A female cannot have a prostate cancer
    - Age should be integer
- **Cancer Message Aggregation (Step 2)**
  - Aggregate relevant cancer messages into one cancer case that contains information of one cancer for one cancer patient
- **Cancer Case Validation (Step 3)**
  - Check validity and correctness of aggregated cancer cases

Cancer Coding Rules

- **Cancer Coding Rules**: core to ensure the activities
  - Breast cancer requires female gender
  - Prostate cancer requires male gender

- A large number: more than **one thousand**
  - Specified by medical chief officers
  - Implemented by medical programmers
  - Applied by medical coders

Cancer Coding Rules

- **Cancer Coding Rules**: core to ensure the activities
  - Breast cancer requires female gender
  - Prostate cancer requires male gender

- A large number: more than **one thousand**
  - Specified by medical chief officers
  - Implemented by medical programmers
  - Applied by medical coders

Challenges in the current practice of CRN

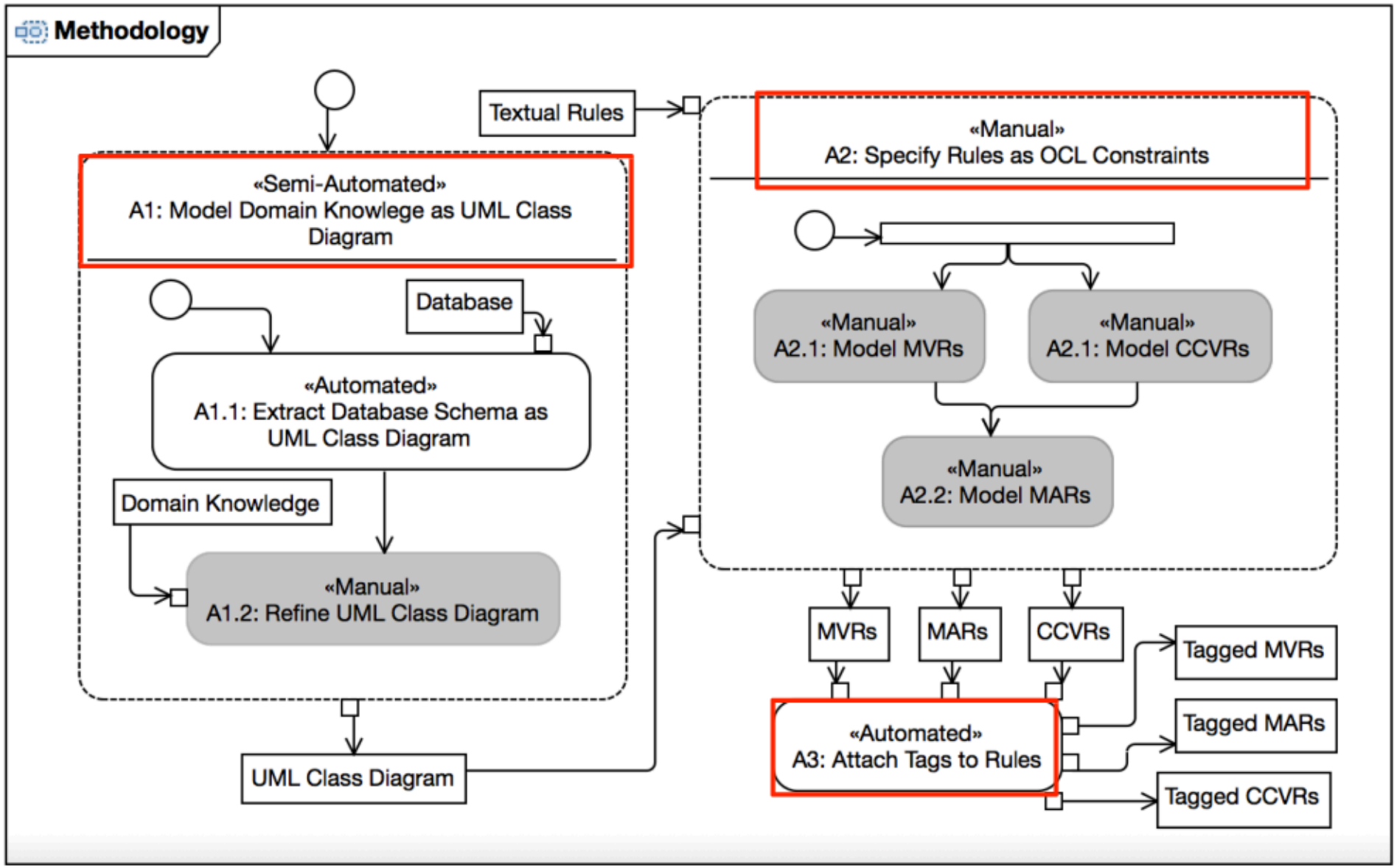
- Low Level of Abstraction
  - Domain knowledge captured in the implementation level
  - Different to learn for fresh stakeholders

- Large Effort for Maintaining Cancer Coding Rules
  - The rules scattered and represented in different ways
  - No unique manner to manage and maintain

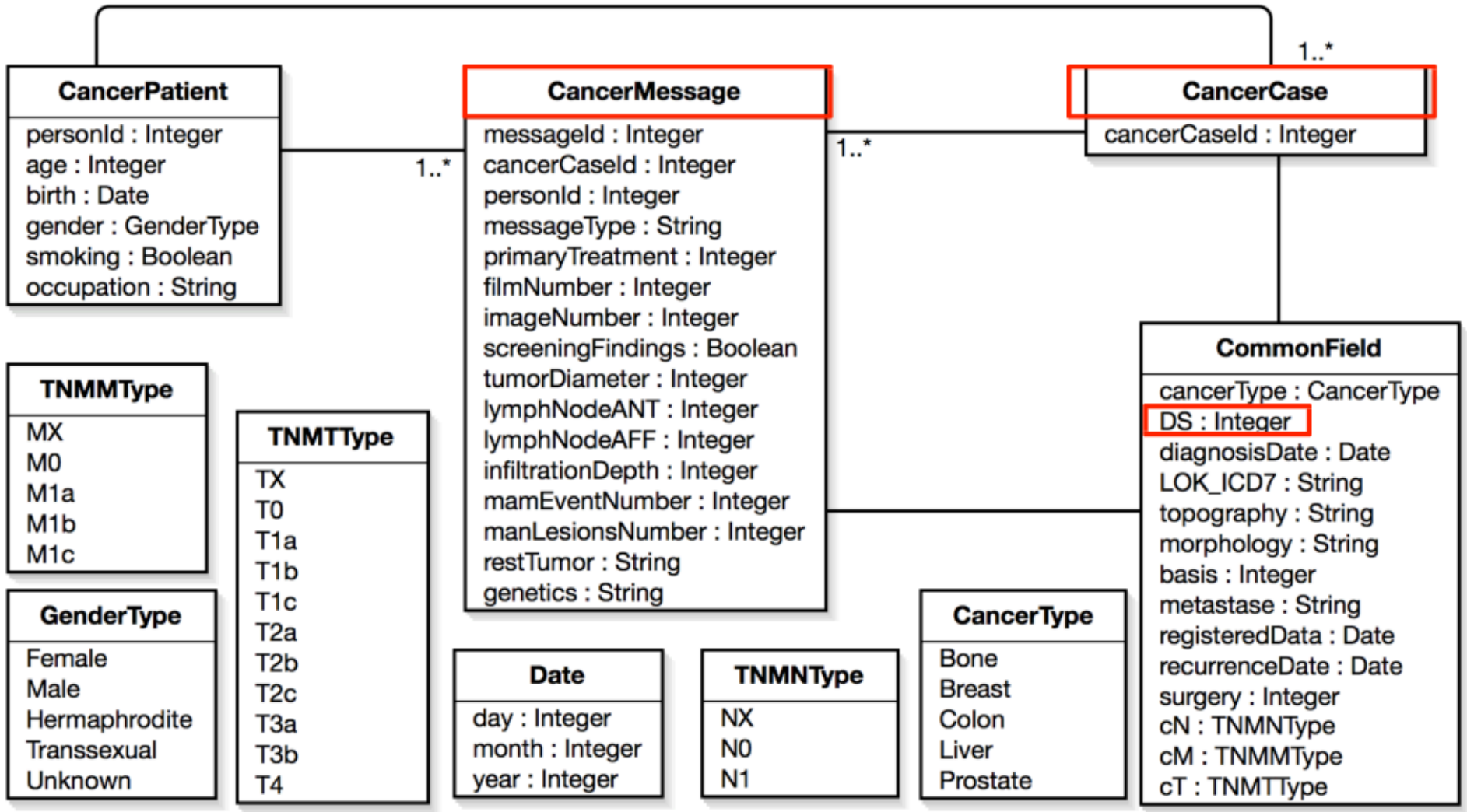
MBF4CR

- **Modeling Methodology**
  - Model the domain knowledge as a UML class diagram
  - Specify cancer coding rules as OCL constraints
  - Associate tags to the OCL constraints
- **Tool Support**

Modeling methodology



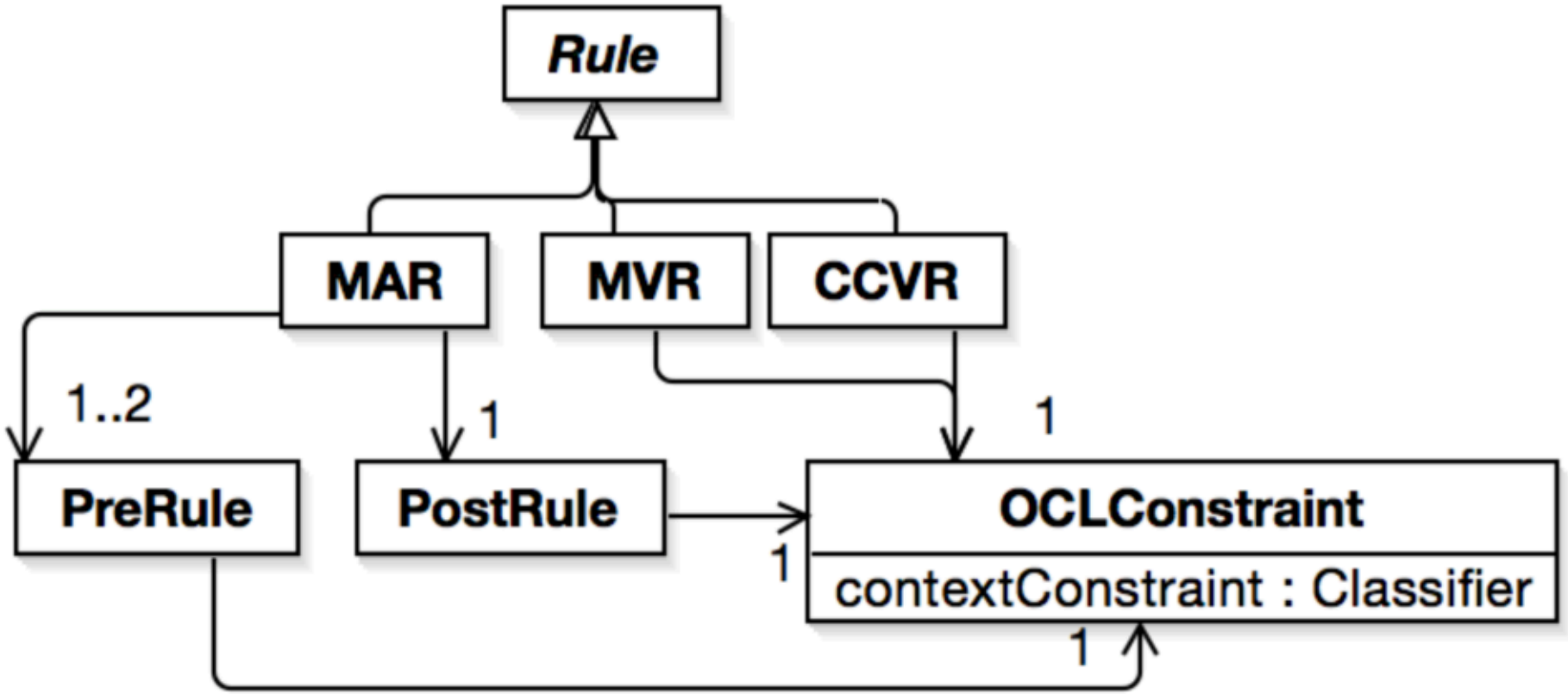
A1: Model the Domain Knowledge as an UML Class Diagram



Statistics

- 64 attributes for *CancerMessage* class
- 49 attributes for *CancerCase* class
- 48 attributes for *CommonField* class

A2: Specify Rules as OCL Constraints



Cancer Message Validation Rules (MVRs)

- Definition
  - Evaluate the validity of one field value of a cancer message
    - DS in a cancer message is an integer ranging from 1 to 9
  - Evaluate the consistency of several fields within a cancer message
    - If the value of attribute basis is equal to “79” and the message type is not “O”, the value of surgery can only choose either “95” or “97”
- Guideline
  - **context** MVR **inv**: self.oclConstraint.contextConstraint = CRN::CancerMessage
- Example
  - **context** CancerMessage **inv**: self.basis = 79 **and** self.messageType <>'O' **implies** (self.surgery = 95 or self.surgery = 97)

Cancer Case validation Rules (CCVRs)

- Definition
  - Evaluate the validity of one field in a cancer case
  - Evaluate the consistency of several fields within a cancer case
    - the value of surgery cannot be 1 when the value of topography is C70, C71 or C72
- Guideline
  - **context** CCVR **inv**: self.oclConstraint.contextConstraint = CRN::CancerCase
- Example
  - **context** CancerCase **inv**: (self.topography = 'C70' **or** self.topography = 'C71' **or** self.topography = 'C72') **implies** self.surgery <>1

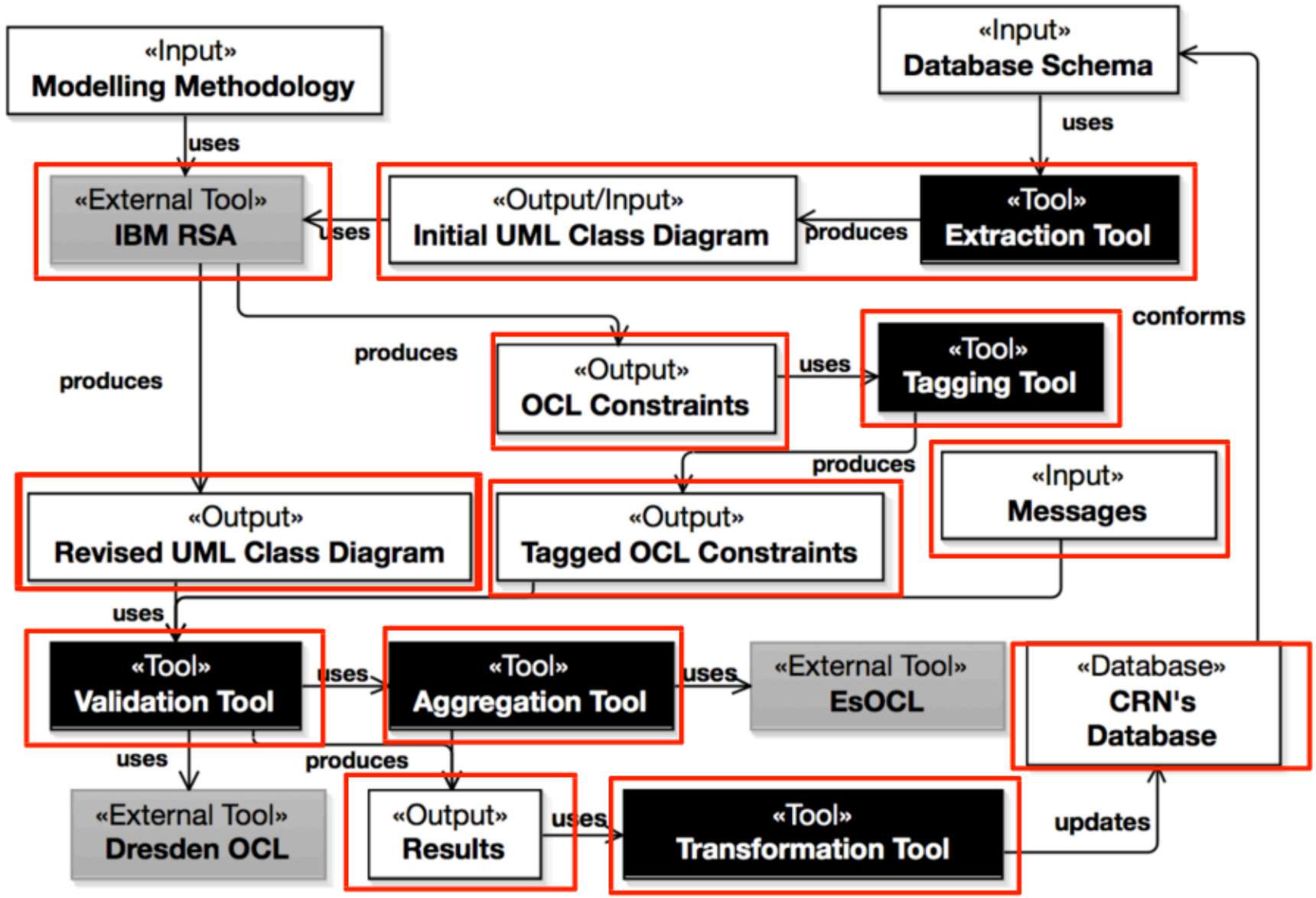
Cancer Message Aggregation Rules (MARs)

- Definition
  - Aggregate one or more cancer messages into a cancer case
    - When a new cancer message comes, if and only if the DS value for the new cancer message is 2 and the current DS value for the cancer case is 3 (**condition**), the DS value for the cancer case will be updated to 2 (**action**)
- Guideline
  - Pre-Rule (Condition)
    - **context** MAR **inv**: self.preRule->size() =1 **implies** self.preRule- >select(preR:PreRule| preR.oclConstraint.contextConstraint = CRN::CancerMessage)->size() =1
    - **context** MAR **inv**: self.preRule->size() =2 **implies** self.preRule.oclConstraint- >select(c:OCLConstraint|c.contextConstraint = CRN::CancerMessage)->size() =1 and self.preRule.oclConstraint->select(c:OCLConstraint|c.contextConstraint = CRN::CancerCase)- >size() =1
  - Post-Rule (Action)
    - **context** PostRule **inv**: self.oclConstraint.contextConstraint = CRN::CancerCase
- Example
  - Pre-rules: **context** CancerMessage **inv**: self.DS=2 **context** CancerCase **inv**: self. DS=3
  - Post-rule: **context** CancerCase **inv**: self.DS=2

A3: Associate Tags to OCL Constraints

- Tags are associated with a rule based on one or more attributes that the rule constrains
- Tags reuse the same names as the attributes in the domain model
- Example
  - **context** CancerMessage **inv**: self.basis = 79 and self.messageType <>'O' **implies** (self.surgery = 95 or self.surgery = 97)
  - Tags: basis, messageType and surgery

Tool Support



Case Study

- A real case study from CRN
  - Cancer data
    - 10 cancer messages
    - 6 cancer cases
  - Medical cancer coding rules
    - 89 MVRs
    - 30 MARs
    - 68 CCVRs

Results-RQ1(Performance)

- Reduce Rule for Execution
  - 32.8% of *MVRs* and 63.3% of *MARs* were selected for the 10 cancer messages
  - 37.5% of CCVRs were selected for all the 6 cancer cases

- Time for Selecting and Executing Rules
  - 732.1, 390.3 and 578 milliseconds for cancer message validation, cancer message aggregation and cancer case validation
  - 25.1, 20.5, 22.7 milliseconds for selecting and executing a MVR, a MAR and a CCVR

Results-RQ2(Correctness)

- Manually checked the results produced by MBF4CR
  - 292 executions of **MVRs**: 879 field check
  - 190 executions of **MARs**: 599 field check
  - 139 executions of **CCVRs**: 958 field check

- Results showed:
  - Correct domain model
  - Correctly specified medical rules
  - Correct implementation

Discussion

- Raising Level of Abstraction and Enabling Automation
  - Facilitate communications of domain knowledge
  - Reduce the cost of training new stakeholders
  - Facilitate the automated process of rule selection
- Systematically Maintaining Medical Rules
  - Only affected OCL constraints need to be updated

Conclusion

- **MBF4CR:** A model-based framework for CRN
  - Systematically model the domain knowledge as UML class diagrams
  - Formally specify different types of medical rules as OCL constraints
  - Associate tags to each OCL constraint to enable an automated rule selection process
- Evaluation with a real case study from CRN
  - Facilitate the current practice with an acceptable performance
  - Comply with the medical domain knowledge
  - Reduce the maintenance effort of medical cancer coding rules

What we have learnt: Take-away

- Global best practice of MBT
- Broad applicability of MBT
- Mature commercial tool support for MBT
- Models are the key factor for success applications of MBT
- MBT for large scale systems and automated testing

Agenda

- Introduction (30 min)
- Start MBT with a simple example (30 min)
- Short break (10 min)
- The value of models (40 min): three real case studies
- **Q&A (10 min)**


Testify
smartere testing